UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR
DEPARTAMENTO DE TEORÍA DE LA SEÑAL Y COMUNICACIONES

IMPLEMENTACIÓN DEL ALGORITMO DE DETECCIÓN DE CARAS DE VIOLA Y JONES SOBRE UNA FPGA

PROYECTO FINAL DE CARRERA

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES: SONIDO E IMAGEN

Autor: Claudio Barroso Heredia

Tutor: Fernando Díaz de María

TRIBUNAL

Presidente:

Secretario:

Vocal:

Realizado el acto de defensa del Proyecto Fin de Carrera el día

de de en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo. Presidente Fdo. Secretario Fdo. Vocal

Agradecimientos

Con este proyecto finaliza una etapa de sacrificio y trabajo, la cual será una de las más importantes de mi vida y donde más he aprendido. Quisiera agradecer a mi madre Francisca, a mi hermano Rubén, a mis abuelos Gregorio y Gloria, y a mi pareja Yolanda todo el apoyo que me han dado siempre, incluso en momentos difíciles. Agradecer también a los compañeros, a mi tutor Fernando Díaz y al resto de docentes de la Universidad Carlos III de Madrid, por todo lo que me han enseñado y ayudado. Y no puedo olvidarme de mi padre Juan, quien ha sido la persona que me ha enseñado a ser como soy y que me ha inculcado la pasión por la tecnología en general. Muchísimas gracias a todos.

Resumen

Este proyecto implementa un sistema de detección de caras basado en el algoritmo de Viola-Jones sobre tecnología FPGA (Field Programmable Gate Array), la cual está cada vez más extendida en ámbitos como las imágenes biomédicas, aplicaciones militares o aeronáutica, entre otros.

El proyecto ha sido desarrollado sobre la placa de desarrollo Altera DE2-115, con un diseño de hardware y software. El hardware consiste en un sistema embebido (SoC) que incluye un procesador de 32 bits que permite ejecutar programas desarrollados en C / C ++, y diferentes módulos permiten la gestión de los distintos componentes. El software ha sido diseñado específicamente para el hardware desarrollado, y permite la detección de rostros en imágenes estáticas; para ello se han desarrollado las clases necesarias para la captura de la imagen una cámara, para la interfaz de usuario, y para la detección de las caras, entre otros.

Palabras clave: FPGA, Viola-Jones, detección de caras.

Abstract

This Project implements a face scanner system based on Viola-Jones algorithm about Field Programmable Gate Array (FPGA) technology, which is becoming widespread in some areas like biomedical images, military or aeronautical applications.

This project has been developed over Altera DE2-115 board, working with both hardware and software designs. The hardware consist in a embedded system (SoC), which includes a 32 bits processor allowing run C / C++ software applications and multiple modules allow management of every single component. The software has been designed specifically for the hardware created, which permits a scanning of faces in static images. For this, classes have been developed for capture images from a camera, user interface and scanner faces, among others.

Keywords: FPGA, Viola-Jones, face detection.

Índice general

Índice de figuras

Índice de tablas

ACRÓNIMOS

A
ABEL: Advanced Boolean Expression Language
ALU: Arithmetic Logic Unit
ASIC: Application-Specific Integrated Circuit

C
CPLD: Complex Programmable Logic Device
CPU: Central Processing Unit

D
DAC: Digital to Analog Converter
DLX: DeLuXe
DMA: Direct Memory Access
DMIPS: Dhrystone Million Instructions Per Second
DRAM: Dynamic Random Access Memory

F
FPGA: Field Programmable Gate Array

G
GPIO: General Purpose I/O

H
HDL: Hardware Description Language

J
JTAG: Joint Test Action Group

L
LCD: Liquid Crystal Display
LE: Logic Element
LED: Light-Emitting Diode

M

MIPS: Million Instructions Per Second
MMU: Memory Management Unit
MPU: Memory Protection Unit

P

PLL: Phase-Locked Loop

R

RAM: Random Access Memory
RISC: Reduced Instruction Set Computer
ROM: Read Only Memory
RTL: Register-Transfer Level
RGB: Red-Green-Blue

S

SOC: System On Chip
SRAM: Static Random Access Memory

V

VGA: Video Graphics Array
VHDL: VHSIC Hardware Description Language
VHSIC: Very-High-Speed Integrated Circuits.
VJ: Viola-Jones

Capítulo 1

Introducción

1.1 Introducción y motivación

Los circuitos integrados están presentes en gran cantidad de productos industriales. Las FPGAs (Field Programmable Gate Array) son una alternativa que poco a poco ha ido introduciéndose en el mercado, a pesar de ser una tecnología que data de 1985 propuesta por Ross Freeman y Bernard Vonderschmitt.

En términos generales, las FPGAs son circuitos integrados de silicio reprogramables. El uso de esta tecnología crece exponencialmente en el mercado gracias a su alto rendimiento y fiabilidad, pero su gran ventaja es la posibilidad de realizar prototipos en fase de desarrollo de sistemas funcionales completos, lo que reduce costes debido a la minimización de los errores en la fase final de producción. Además permite la posibilidad de, una vez en el mercado, actualizar completamente el sistema remotamente si lo requiriera, característica permitida solo por este tipo de tecnología junto con su antecesor, los CPLDs (Complex Programmable Logic Device).

El uso de las FPGAs está creciendo gracias a su robustez y fiabilidad. Por ello están reemplazando en el mercado a los denominados ASICs (Application-Specific Integrated Circuit) gracias a las ventajas de la tecnología FPGA frente a la tecnología ASIC que se muestran en las siguientes tablas [1]:

FPGAs

Ventaja	Beneficio
Breve tiempo de salida al mercado	No máscaras u otros pasos de fabricación necesarios.
No hay gastos no recurrentes por adelantado	Costes usualmente asociados a los diseños ASIC
Fase de diseño más simple	Debido a que el software de desarrollo maneja la mayor parte del enrutamiento, la colocación y los tiempos del circuito.
Programabilidad remota	Permite reprogramar el sistema íntegro remotamente.

Tabla 1.1 Ventajas y beneficios de la tecnología FPGA.

ASICs

Ventaja	Beneficio
Máxima capacidad de personalización	Los dispositivos ASICs son diseñados íntegramente para una aplicación específica.
Costes unitarios más bajos	Costos más bajos para pedidos a gran escala.
Integrado físicamente de menor tamaño	Debido a que es diseñado para una aplicación específica.

Tabla 1.2 Ventajas y beneficios de la tecnología ASIC.

Prueba del incremento en el mercado de las FPGAs son las aplicaciones en aeronáutica, robótica, procesamiento de imágenes médicas, entre otros. A continuación se muestran algunos ejemplos de tales aplicaciones (terminados o en fase de desarrollo):

Medicina y asistencia:

- Implementación de un sistema de control para brazos robóticos asistenciales basado en tratamiento de imagen. Universidad de Florida en conjunto con Mitsubishi [2].

Militar:

- Digitalizador de balística. Bittware [3].
- Sistema radar para aeronaves furtivos o indetectables al radar. Bittware [4].

El desarrollo sobre FPGAs se realiza, al igual que en la tecnología ASIC, en lenguaje de programación HDL. Este lenguaje es de bajo nivel. La programación a este nivel permite al desarrollador la libertad total de diseño, pero por contra, requiere mayor esfuerzo y coste para la empresa desarrolladora. Por ello una alternativa es desarrollar en lenguajes de programación de más alto nivel, es decir, con mayor capacidad de abstracción como C/C++ o Java. Esto se consigue implementando en la propia FPGA los denominados SoC (System On Chip) que integran un sistema completo compuesto por uno o varios procesadores, bloques de memoria, controladores para interfaces (como USB o Ethernet) o DSPs (Digital Signal Process), entre otros. Los sistemas SoC hacen más fácil el desarrollo de aplicaciones específicas ya que permiten la posibilidad de escribir el software en el lenguaje de programación C/C++, en lugar de en el lenguaje de bajo nivel VHDL/Verilog.

Con el fin de experimentar con una herramienta de desarrollo sobre FPGAs y verificar sobre una aplicación real las capacidades, ventajas y limitaciones de este tipo de

tecnología, en este trabajo nos proponemos implementar en FPGA un algoritmo que tenga interés en la actualidad y que nos permita valorar adecuadamente la herramienta. En particular nos hemos decantado por la implementación de un sistema de detección de caras.

La detección de caras está cada vez más presente en la industria gracias al enorme abanico de aplicaciones prácticas (monitorización de individuos, identificación biométrica, etc), implementadas en aplicaciones de redes sociales, de videovigilancia, o de defensa, entre otros. Debido al alto interés de la industria en estas aplicaciones han surgido diversos métodos y algoritmos ([5], [6]) de los que cabe destacar el algoritmo desarrollado por Paul Viola y Michael Jones [7]. El algoritmo de VJ (Viola-Jones) ha obtenido unos resultados extraordinarios (alta tasa de detección y baja posibilidad de falsa alarma) con un coste computacional muy reducido. Además no es exclusivo para la detección de caras, ya que puede ser implementado en cualquier aplicación donde se necesite detectar objetos con variabilidad estructural en tiempo real.

1.2 Objetivos

El objetivo principal de este proyecto es el desarrollo de una aplicación de detección de caras; sobre una placa de desarrollo que incorpora una FPGA interconectada a una salida VGA (Video Graphics Array), a una entrada de cámara, y a diversos módulos de memoria entre otros.

El proyecto está dividido en dos partes:

- Desarrollo del hardware del sistema.
- Desarrollo del software de detección de caras.

El sistema completo junto con la aplicación de detección de caras estará principalmente enfocado a optimizar la carga computacional y minimizar los tiempos de detección, debido a las limitaciones de los recursos utilizados. Las líneas de desarrollo del prototipo son las siguientes:

Hardware:

- Creación de un sistema completo (SoC) que incluirá entre otros un procesador embebido con capacidad de ejecutar software escrito en C/C++.

- Optimización de los recursos de almacenamiento de datos. Distribución de los módulos de memoria de diferentes tecnologías (Flash, SDRAM y SRAM).

Software :

- Desarrollo de la interfaz gráfica y clases para tratamiento de imágenes.

- Desarrollo de la gestión VGA, eliminación de parpadeos con el método de doble buffer.

- Implementación del algoritmo de VJ de detección de caras.

-Optimización eficiente de los parámetros del algoritmo de VJ atendiendo el compromiso entre tiempos de detección y probabilidad de detección, para posibilitar la implementación en tiempo real.

1.3 Estructura de la memoria

Capitulo 2 - Diseño Hardware

- Estado del Arte (apartado 2.1): En este apartado se especifican los conceptos básicos para la compresión de los componentes empleados en un sistema SoC, en el que se incluyen conceptos relacionados con los procesadores embebidos.

- Diseño e implementación (apartado 2.2). En este apartado se describen los pasos seguidos para desarrollar el sistema embebido (SoC) sobre la FPGA.

Capitulo 3 - Implementación del algoritmo de detección de caras

- Estado del Arte (apartado 3.1): En este apartado se especifican los conceptos básicos para la compresión del algoritmo de Viola-Jones para la detección de caras.

- Implementación software sobre la FPGA del algoritmo de Viola-Jones (apartado 3.2): En este apartado se desarrolla el software de detección de caras implementado sobre el SoC.

Capitulo 4 - Resultados de la evaluación

En este capítulo se valoran los resultados obtenidos de las pruebas realizadas para la mejora del compromiso entre tiempo de detección y probabilidad de detección, mediante el manejo de parámetros en la implementación del algoritmo de Viola-Jones.

Capitulo 5 - Conclusiones y líneas futuras

En el capítulo 5 se resumen las conclusiones extraídas de los resultados obtenidos en el desarrollo del proyecto. Se hace referencia a posibles mejoras para el prototipo, incluyendo mejoras para la interfaz gráfica, mejoras para el rendimiento de la detección de caras y mejoras relacionadas con el hardware.

Capitulo 6 - Presupuesto

En este capítulo se detalla el presupuesto del proyecto.

Capítulo 2

Diseño Hardware

2.1 Estado del Arte

2.1.1 FPGA

Las FPGAs son dispositivos digitales programables de propósito general compuestos por bloques lógicos interconectados que surgen como resultado de la unión entre las tecnologías PLD y PAL (“Programmable Array Logic”). Esta tecnología fue presentada en 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx.

Las FPGAs permiten desarrollar mediante lógica programable, desde funciones sencillas como una puerta lógica, hasta sistemas combinacionales complejos como los denominados SoC (“system-on-chip”) los cuales son sistemas completos con una funcionalidad y estructura similar a la de un ordenador (arquitectura de John von Neumann [8]).

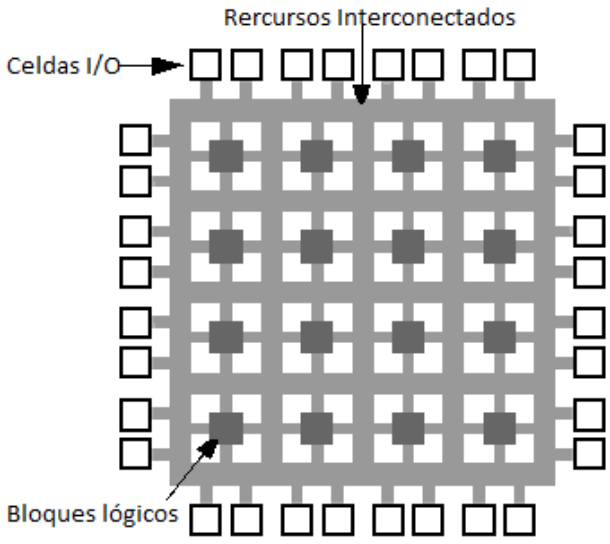


Figura 2.1 Arquitectura interna de una FPGA.

El número de bloques programables varía dependiendo de la arquitectura y estructura pudiendo llegar a los 8 millones de estas celdas lógicas, lo que garantiza la implementación prácticamente de cualquier sistema o aplicación.

Los principales fabricantes en la actualidad son Xilinx y Altera con arquitecturas y prestaciones muy similares, y son configurados mediante un lenguaje de descripción de hardware, generalmente Verilog, VHDL o ABEL. Recientemente se están implementando los denominados SoC ("System-on-Chip"), los cuales ya estaban integrados en sus predecesores PAL. Estas plataformas o módulos son sistemas embebidos dentro de la propia FPGA que tienen la ventaja de encapsular procesadores, memorias, interfaces para periféricos externos (incluidas memorias externas) y las diversas conexiones entre éstos (buses). Todos los módulos de estos sistemas son muy configurables (incluido el microprocesador), pudiendo configurar instrucciones personalizadas, incluir memoria cache de instrucción y datos, MMU ("Memory Management Unit") o MPU ("Memory Protection Unit"), entre otros.

2.1.1.1 Xilinx

Xilinx, Inc. es la compañía líder en el desarrollo de FPGAs. Fue fundada en 1984 por Ross Freeman, Bernard Vonderschmitt y Jim Barnett y tiene su sede en San José, California, perteneciente a la zona denominada *Silicon Valey.* En la actualidad su objetivo de mercado está centrado principalmente en el desarrollo de dispositivos FPGA y CPLD.

En lo referente a las FPGAs, Xilinx dispone de un amplio abanico de soluciones, que incluyen desde dispositivos de alto rendimiento como la serie Virtex, hasta modelos de bajo coste como la serie Spartan.

Las características de los diferentes modelos se presentan de forma resumida en la siguiente tabla comparativa [9]:

	Spartan-6	**Artix-7**	**Kintex-7**	**Virtex-7**	**Kintex UltraScale**	**Virtex UltraScale**
Celdas Lógicas	147.443	215.360	477.760	1.954.560	1.160.880	4.407.480
Memoria Embebida	4,8 Mb	13 Mb	34 Mb	68 Mb	76 Mb	115 Mb
DSP Slices	180	740	1.920	3.600	5.520	2.880

Rendimiento DSP	140 GMACs	930 GMACs	2.845 GMACs	5.335 GMACs	8.180 GMACs	4.268 GMACs
Número de transceptores	8	16	32	96	64	104
Velocidad de transceptor	3,2 Gb/s	6,6 Gb/s	12,5 Gb/s	28,05 Gb/s	16,3 Gb/s	32,75 Gb/s
Ancho de banda total del transceptor *(full duplex)*	50 Gb/s	211 Gb/s	800 Gb/s	2.784 Gb/s	2.086 Gb/s	5.101 Gb/s
Interfaz de memoria DDR3	800	1.066	1.866	1.866	2.400	2.400
Interfaz PCI Express®	x1 Gen1	x4 Gen2	x8 Gen2	x8 Gen3	x8 Gen3	x8 Gen3
Señal mixta analógica (AMS)/XADC	-	XADC	XADC	XADC	Monitor de sistema	Monitor de sistema
Configuración AES	Si	Si	Si	Si	Si	Si
Pines de usuario I/O	576	500	500	1.200	832	1.456
Voltaje de pines	1,2V - 3,3V	1,2V - 3,3V	1,2V - 3,3V	1,2V - 3,3V	1,0V - 3,3V	1,0V - 3,3V

Tabla 2.1 Comparativa de dispositivos FPGAs de Xilinx.

Los entornos de desarrollo de Xilinx para los dispositivos FPGA son las siguientes:

- **ISE Design Suite**: Este conjunto de aplicaciones de desarrollo permite el análisis y síntesis de diseños programados en lenguaje de descripción de hardware (HDL), permitiendo además el análisis de tiempos simulados y examinar diagramas RTL (“Register-Transfer Level”). En lo referente a sistemas embebidos, Xilinx proporciona el sistema MicroBlaze, que permite la ejecución de programas o aplicaciones desarrollados en lenguajes de alto nivel como C. Las herramientas a destacar incluidas en este entorno son las siguientes:

  - “ISE Project Navigator”: Herramienta principal para diseños desarrollados en HDL.

- "Plataform Studio": Herramienta para el desarrollo de sistemas embebidos MicroBlaze.
- "Software Development Kit" (SDK): Herramienta para el desarrollo de aplicaciones sobre los sistemas embebidos MicroBlaze.

- **Vivado Design Suite**: Este entorno es la evolución de ISE Design Suite para el diseño de sistemas sobre las FPGAs de la serie 7 de los modelos Artix, Virtex y Kintex. Las aplicaciones más destacadas son las siguientes:

  - Vivado IDE.
  - Software Development Kit.
  - Vivado Integrated Design Enviroment for MicroBlaze.

2.1.1.2 Altera

Altera Corporation es, junto con Xilinx, el otro gran líder del mercado en lo referente a dispositivos FPGA. Fue fundada en 1983 y desarrolló su primer PLD en 1984. Al igual que Xilinx, tiene establecida su sede en San José, California. El objetivo de mercado de Altera son las FPGAs, CPLDs y ASICs, por lo que es el competidor directo de Xilinx. En el ámbito de las FPGAs, Altera proporciona modelos de bajo coste (serie Cyclone), modelos de rendimiento medio (serie Arria) y modelos de rendimiento alto (serie Stratix).

Las características de los diferentes modelos se presentan de forma resumida en la siguiente tabla comparativa [10].

	Cyclone III EP3CLS100F484C8N	**Cyclone IV** EP4CE115F29C7	**Arria V** 5AGXMA3D4F27C4N	**Stratix IV** EP4SGX230KF40C2N
Elementos Lógicos (LEs)	100.500	114.500	156.000	228.000
Memoria embebida	4.347 Kbits	3.800 Kbits	10.510 Kbits	14.283 Kbits
Bloques de memoria M9K (256x36 bits)	483	432	0	1.235
Multiplicadores 18x18 bits	267	266	792	1.288
Máximo rendimiento DSP	200 MHz	250 MHz	370 MHz	600 MHz

Número de transceptores	0	0	9	36
Velocidad de transceptor	-	-	6,554 Gbps	8,5 Gbps
Interfaz PCI Express®	-	-	x1	x2
Interfaz DDR3	-	-	533 Mbps	1.067 Mbps
Interfaz DDR2	333 Mbps	200 Mbps	400 Mbps	800 Mbps
PLLs	4	4	10	8
Pines de usuario I/O	278	528	336	736
Voltaje de pines	2,5V - 3,3V	1,2V - 3,3V	1,2V - 3,3V	2,5V - 3,3V

Tabla 2.2 Comparativa de dispositivos FPGAs de Altera.

Por otro lado, las herramientas de Altera para el desarrollo son las siguientes:

- **Quartus II**: Quartus II es el entorno de desarrollo principal que proporciona Altera para el análisis y síntesis de diseños en lenguaje HDL, como Verilog, VHDL y ABEL. Al igual que su equivalente de Xilinx (ISE), permite el análisis de tiempos, la simulación de los diseños y la visualización de los esquemas RTL correspondientes. Existen dos versiones de Quartus II, una gratuita denominada Quartus II Web Edition y una versión de pago denominado Quartus II Subscription Edition. Las diferencias entre estas versiones no limitan el desarrollo completo de un sistema cualquiera, pero sí limitan características en la implementación final sobre la FPGA, como por ejemplo la restricción de tener conectado permanentemente el equipo de desarrollo con la FPGA durante la ejecución en la versión gratuita. Además de estas diferencias, la versión gratuita no soporta todos los modelos existentes CPLD y FPGA, mientras que la versión de pago soporta todos los modelos.

- **SOPC Builder**: Esta aplicación permite la creación de un sistema embebido completo, incluyendo los correspondientes ficheros de código HDL para ser implementados en Quartus II. Proporciona los componentes necesarios para realizar el sistema, incluyendo el procesador Nios II de Altera.

- **NIOS II Embedded Design Suite (EDS)**: EDS es un conjunto de herramientas con el objetivo de implementar programas escritos en C/C++ y en lenguaje ensamblador sobre un sistema embebido con procesador NIOS II. Su entorno de desarrollo esta basado en Eclipse, el cual es un entorno de código abierto multilenguaje.

2.1.2 Procesadores embebidos

Los procesadores embebidos o *soft-processors* son procesadores que pueden implementarse en una FPGA, CPLD o ASIC mediante puertas lógicas. Estos procesadores pueden tener varios modelos más complejos dependiendo del número de bloques lógicos de la FPGA correspondiente, e incluso también es posible implementar más de un procesador, lo que permitiría realizar tareas en paralelo. La información de las tablas de rendimiento de los procesadores de esta sección está expresada en DMIPS (Dhrystone Millones de Instrucciones Por Segundo), basado en el algoritmo de Dhrystone que centra su análisis en la ALU (“Arithmetic Logic Unit”) del procesador.

2.1.2.1 MicroBlaze de Xilinx

Xilinx propone MicroBlaze, un procesador de 32 bits que puede ser implementado en cualquier FPGA de este fabricante. Su arquitectura interna es muy similar a RISC (“Reduced Instruction Set Computer”), basado en DLX (DeLuXe) diseñado por John L. Hennessy y David A. Patterson [11]. Este procesador permite implementar instrucciones personalizadas que generalmente no son utilizadas en aplicaciones habituales, como instrucciones de multiplicación y división, u operaciones en variables de punto flotante. MicroBlaze contiene más de 70 opciones configurables por el usuario, entre las que destaca la configuración del tamaño de la memoria caché, la gestión de los periféricos integrados (como los PLL) y la unidad de gestión memoria (MMU), entre otros. En la figura 2.2 se muestra el diagrama de bloques del procesador MicroBlaze [12].

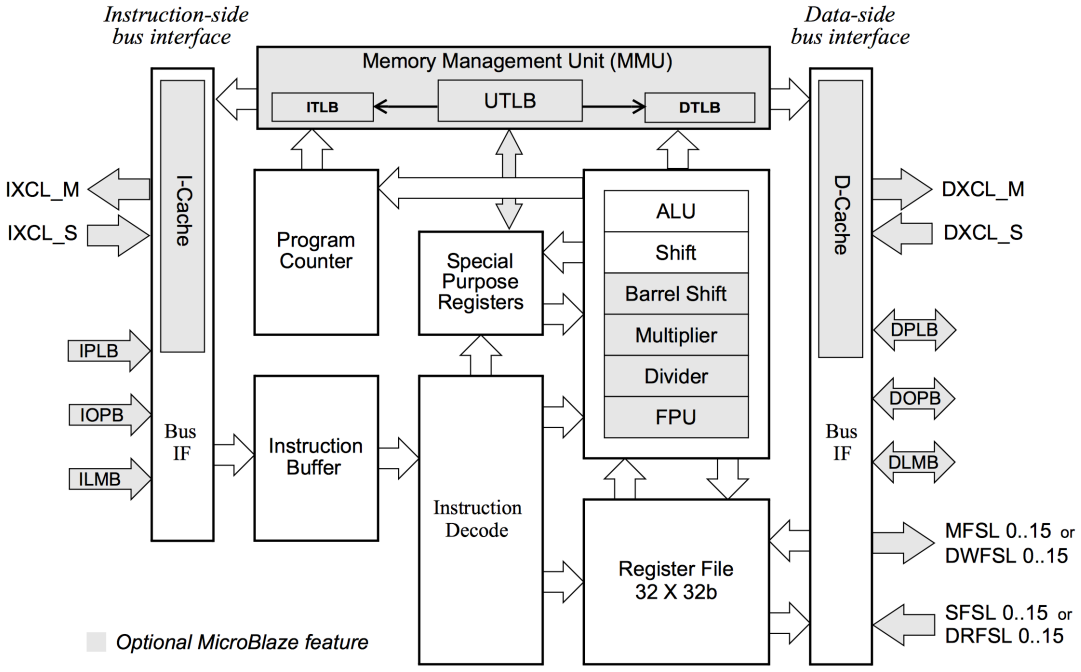


Figura 2.2 Diagrama de bloques del procesador MicroBlaze.

La siguiente tabla compara el rendimiento del procesador MicroBlaze implementado en diferentes FPGAs de Xilinx [13]:

Familia de FPGA	**Rendimiento optimizado de MicroBlaze con optimización del predictor de saltos (*branch*)** **5 estados de segmentación (*pipeline*)**	**Rendimiento optimizado de MicroBlaze** **5 estados de segmentación (pipeline)**	**Área optimizada MicroBlaze** **3 estados de segmentación (pipeline)**
Virtex-7 FPGA	293 DMIPs	393 DMIPs	264 DMIPs
Kintex-7 FPGA	317 DMIPs	408 DMIPs	264 DMIPs
Virtex-6 FPGA	306 DMIPs	384 DMIPs	246 DMIPs
Spartan-6 FPGA	166 DMIPs	209 DMIPs	152 DMIPs

Tabla 2.3 Comparativa de rendimiento de MicroBlaze en diferentes dispositivos.

El desarrollo de aplicaciones para el procesador MicroBlaze se realiza en el entorno EDK (“Embedded Development Kit”) basado en Eclipse, en lenguaje C/C++.

2.1.2.2 Nios II de Altera

Nios II es el procesador embebido para productos Altera. Se trata de un procesador de 32 bits, evolución directa de su predecesor Nios de 16 bits. Su arquitectura interna está basada en RISC realizada en su totalidad por bloques lógicos. Al igual que MicroBlaze, Nios II permite al usuario configurar múltiples opciones como el tamaño de la memoria cache, instrucciones personalizadas de hasta 32 bits, la unidad de gestión de memoria (MMU) y la unidad de protección de memoria (MPU), entre otros [14]. La figura 2.3 muestra el diagrama de bloques del procesador Nios II [15]:

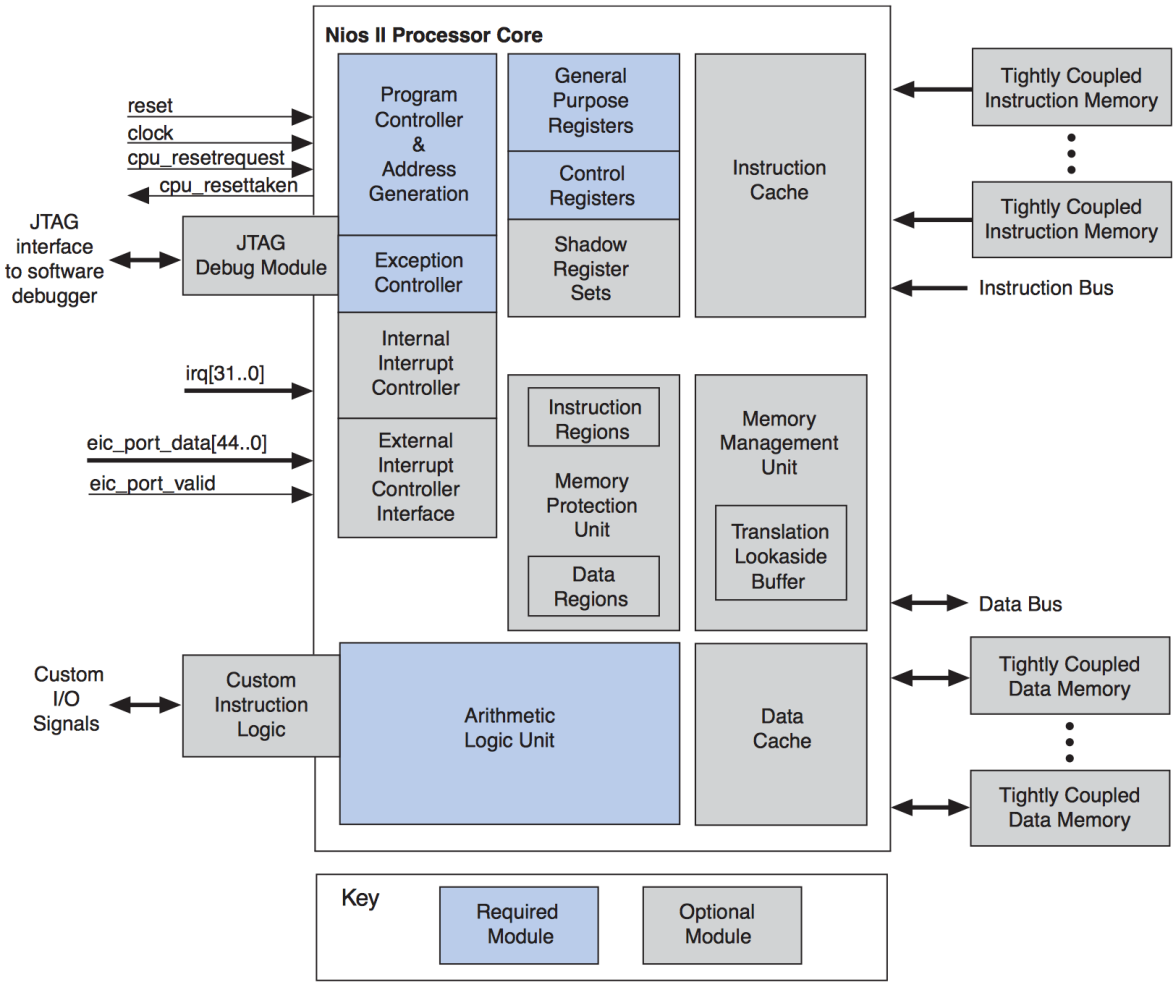


Figura 2.3 Diagrama de bloques del procesador Nios II.

Nios II proporciona tres modelos diferenciados en su rendimiento general, según el número de recursos utilizados en la FPGA. A continuación se describen las características de dichos modelos [16]:

- **Nios II/e Core**: Nios II/e o Nios II *economy* es el modelo del procesador Nios II que menor número de recursos consume (aproximadamente 700 LEs), a cambio de un menor rendimiento. Su rendimiento es de aproximadamente 30 DMIPS a velocidades de hasta 200 MHz. Las características principales del procesador Nios II/e son las siguientes:

    - El acceso a un máximo de 2 GB de espacio de direcciones externo.
    - Módulo de depuración JTAG ("Joint Test Action Group").
    - Mejoras de depuración opcionales.
    - Soporte de hasta 256 instrucciones personalizadas.

- **Nios II/s Core**: Nios II/s o Nios II *standard* es el modelo diseñado para maximizar un rendimiento y un consumo de recursos equilibrado. Este procesador es óptimo para aplicaciones de medio rendimiento sensibles a los costes, incluidas las aplicaciones que procesan gran cantidad de datos como un sistema operativo completo. Las características principales del procesador Nios II/s son las siguientes:

    - Caché de instrucciones.
    - El acceso a un máximo de 2 GB de espacio de direcciones externo.
    - Segmentación de cinco etapas.
    - Opción de división y multiplicación por hardware.
    - Soporte de hasta 256 instrucciones personalizadas.
    - Modulo de depuración JTAG (Joint Test Action Group).
    - Mejoras del módulo de depuración como puntos de interrupción de hardware, disparadores de datos y monitorización en tiempo real.

- **Nios II/f Core**: Nios II/f o Nios II *fast* es el modelo del procesador Nios II diseñado específicamente para un alto rendimiento. Con un rendimiento de 300 MIPS (referenciados en Dhrystones), es óptimo para aplicaciones de rendimiento crítico, así como aplicaciones que hagan uso de grandes cantidades de código y datos, tales como un sistema operativo con todas las funcionalidades disponibles. Las características principales del procesador Nios II/f son las siguientes:

    - Unidad de gestión de memoria (MMU).
    - Unidad de protección de memoria (MPU).
    - Controlador de interrupciones vectorizadas externa.
    - Soporte avanzado de excepciones.
    - Caché de instrucciones y datos separadas (512 bytes y 64 KB).

- El acceso a un máximo de 2 GB de espacio de direcciones externo.
- Segmentación de seis etapas.
- Opción de división por hardware.
- Soporte de hasta 256 instrucciones personalizadas.
- Modulo de depuración JTAG (Joint Test Action Group).
- Mejoras del modulo de depuración como puntos de interrupción de hardware, disparadores de datos y monitorización en tiempo real.

2.1.3 Memorias de almacenamiento

Las memorias de almacenamiento son dispositivos que retienen datos digitales durante un intervalo de tiempo. Es uno de los componentes fundamentales de las computadoras, que incorporado a un sistema con una CPU o procesador, implementa la arquitectura fundamental de John von Neumann, la cual es la base de las computadoras actuales. Al ser de naturaleza digital, almacena únicamente datos binarios (‘1’ o ‘0’), denominados bits. En el ámbito de la computación, la unidad de medida más utilizada es el byte, que equivale a 8 bits. Esto es así debido a que el procesamiento de datos en las computadoras se realiza en múltiplos de 8 bits. Por ejemplo, los procesadores embebidos descritos en el apartado 2.2 procesan información en tramas de 32 bits, o su equivalente 4 bytes.

Además del byte se utilizan unidades mayores para facilitar su manejo; a continuación se muestra una tabla de equivalencia de las unidades más utilizadas en el ámbito de la computación:

Nombre	Símbolo	Número de bytes	Equivalencia
bit	b	1/8	-
byte	B	1	8 bits
kilobyte	KB	1.024	1.024 B
megabyte	MB	1.048.576	1.024 KB
gigabyte	GB	1.073.741.824	1.024 MB
terabyte	TB	1.099.511.627.766	1.024 GB

Tabla 2.4 Tabla de conversión binaria.

En los siguientes apartados de esta sección se introducirán algunas de las denominadas memorias de estado sólido, es decir, memorias sin ningún mecanismo móvil las cuales son utilizadas en este proyecto.

2.1.3.1 Memorias de acceso aleatorio volátiles (RAM)

Las memorias RAM ("Random Access Memory") son memorias que permiten almacenar cantidades relativamente grandes de datos. Este tipo de memorias permiten realizar dos tipos de operaciones sobre los datos:

- **Lectura**: Esta operación permite la recuperación de datos que han sido almacenados previamente mediante una operación de escritura.
- **Escritura**: Mediante esta operación se proporcionan datos a la memoria para su posterior recuperación mediante una operación de lectura.

Estas operaciones no se efectúan bit a bit, sino que se realizan en bloques de *N* bits consecutivos denominados "palabras" (*words*). Cada una de estas palabras están numeradas en la memoria para su localización. Esta asignación se denomina "dirección" (*address*), permitiendo así la lectura/escritura de dicha palabra indicando este parámetro asociado.

Las señales de control y datos de este tipo de memorias se enumeran a continuación:

- **Bus de datos**: Es el bus principal de la memoria junto con el bus de direcciones, y permite la entrada de datos para el proceso de escritura y la salida de datos para el proceso de lectura. Debido a que permite la doble dirección de los datos, este tipo de buses se denominan bidireccionales.

- **Bus de direcciones**: Mediante este bus se indica la dirección de memoria a la que desea acceder, ya sea para escritura o para lectura. El ancho de este bus depende del número de palabras de que conste la memoria.

- **Bus de control**: Este bus es un conjunto de señales que se utilizan para realizar los accesos a la memoria, incluido el tipo de operación.

Dentro de la tecnología RAM, existen dos variantes diferenciadas, las memorias SRAM ("Static Random Access Memory") y las memorias DRAM ("Dynamic Random Access Memory") que se introducen en los siguientes apartados.

2.1.3.1.1 Memorias SRAM

Las memorias estáticas SRAM (“Static Random Access Memory”) son un tipo de memorias RAM que mantienen la información mientras se mantenga la tensión de alimentación. Las señales de control más habituales en este tipo de memorias son las siguientes:

- **Habilitación de lectura/escritura (/WE o /RW)**: Esta señal indica el tipo de operación a realizar, escritura o lectura.

- **Selección de chip (/CS o /CE)**: Esta señal permite habilitar o deshabilitar el control de la memoria, permitiendo que otros circuitos integrados usen el mismo bus del sistema.

- **Habilitación de salida (/OE)**: Esta señal controla el triestado de las salidas, permitiendo ponerlas en alta impedancia (‘Z’).

En la figura 2.4 se muestran las señales de control, datos y dirección de las memorias estáticas RAM [17].

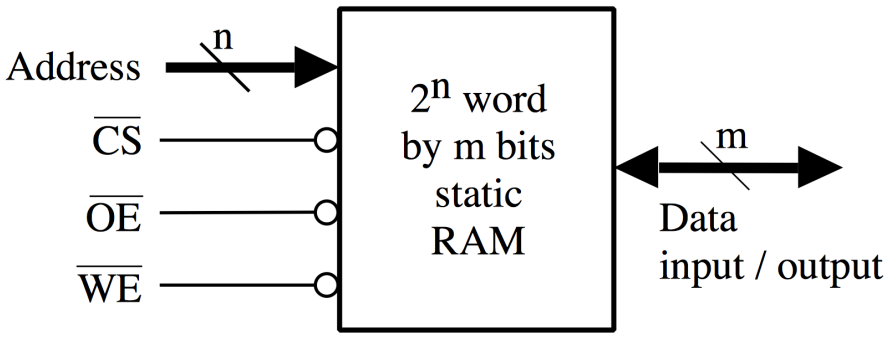


Figura 2.4 Señales de las memorias SRAM.

Las siguientes figuras 2.5 y 2.6 muestran los diagramas de tiempos para las operaciones de lectura y escritura respectivamente, en las memorias SRAM [18]:

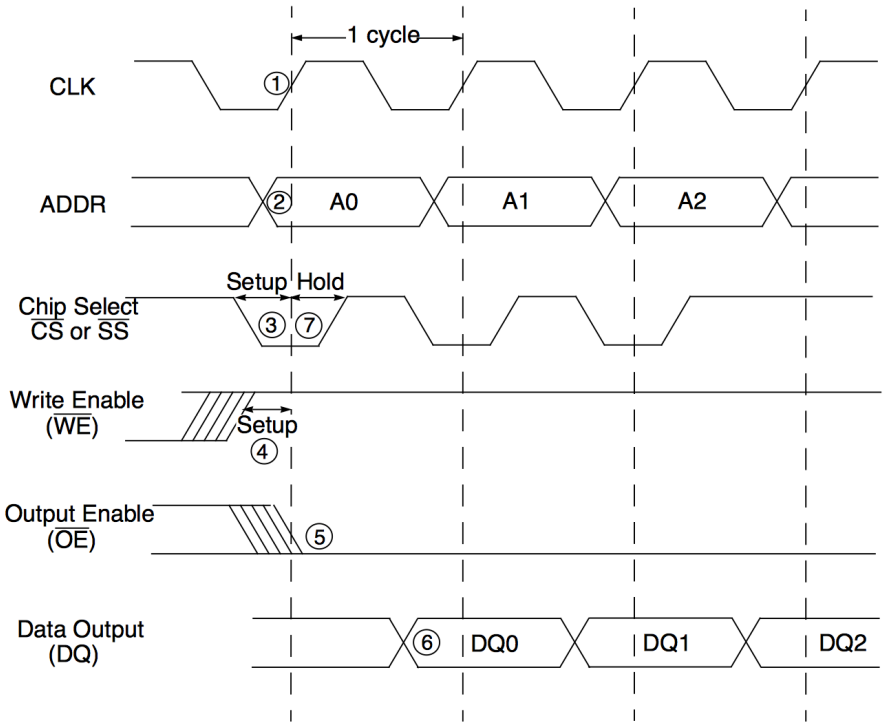


Figura 2.5 Diagrama de tiempos para la lectura de datos en memorias SRAM.

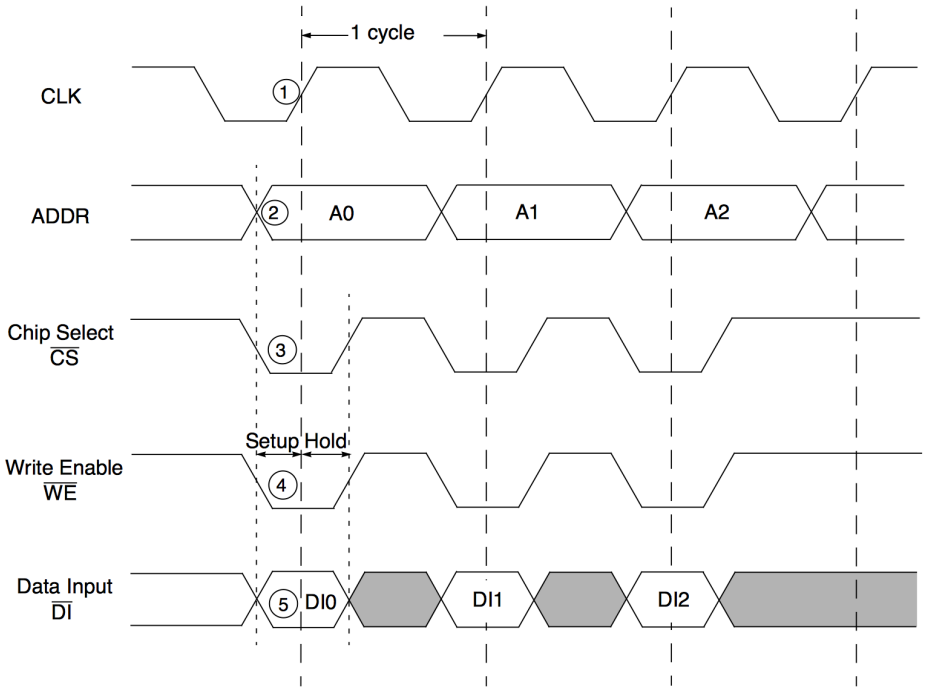


Figura 2.6 Diagrama de tiempos para la escritura de datos en memorias SRAM.

Las ventajas de las memorias SRAM son la alta velocidad de acceso y el bajo consumo eléctrico debido al mayor aislamiento de las celdas de datos, evitando el refresco continuo de las memorias DRAM. El inconveniente de este aislamiento, es el mayor tamaño físico de las celdas, y por ello la menor capacidad comparadas con las memorias DRAM.

2.1.3.1.2 Memorias DRAM

Las memorias DRAM ("Dynamic Random Access Memory") se diferencian de las memorias SRAM en la estructura de las celdas. Mientras que en las memorias SRAM son necesarios varios transistores por celda para aislar totalmente la celda del exterior a costa de un mayor tamaño físico, las celdas de las memorias DRAM constan de menos transistores por celda con el inconveniente de un menor aislamiento pero con la ventaja de permitir una mayor capacidad. Al tener menor capacidad de aislamiento es necesario refrescar las celdas, es decir, reescribir periódicamente su contenido, lo que se traduce en menor velocidad de acceso y mayor consumo eléctrico.

Junto con la mayor capacidad de almacenamiento aparece el problema del número de pines necesarios para direccionar dicha memoria. Este aumento de pines hace que el integrado sea mayor y por lo tanto más costoso de utilizar. Para el control del número de pines, se divide la dirección en dos partes: filas y columnas. Este mecanismo consigue reducir el número de pines de dirección a la mitad, a cambio de incrementar el tiempo de acceso.

Las señales de control de las memorias DRAM se detallan a continuación:

- **Habilitación de columna (/CAS)**: Controla el almacenamiento de la parte de la dirección correspondiente a la columna.

- **Habilitación de fila (/RAS)**: Controla el almacenamiento de la parte de la dirección correspondiente a la fila.

- **Habilitación de lectura/escritura (/WE o /RW)**: Esta señal indica si se procede a realizar una operación de lectura o escritura sobre la memoria.

- **Habilitación de salida (/OE)**: Esta señal controla el triestado de las señales, permitiendo ponerlas en alta impedancia para la compartición del bus.

La figura 2.7 muestra el diagrama de bloques de la arquitectura de las memorias DRAM [19]:

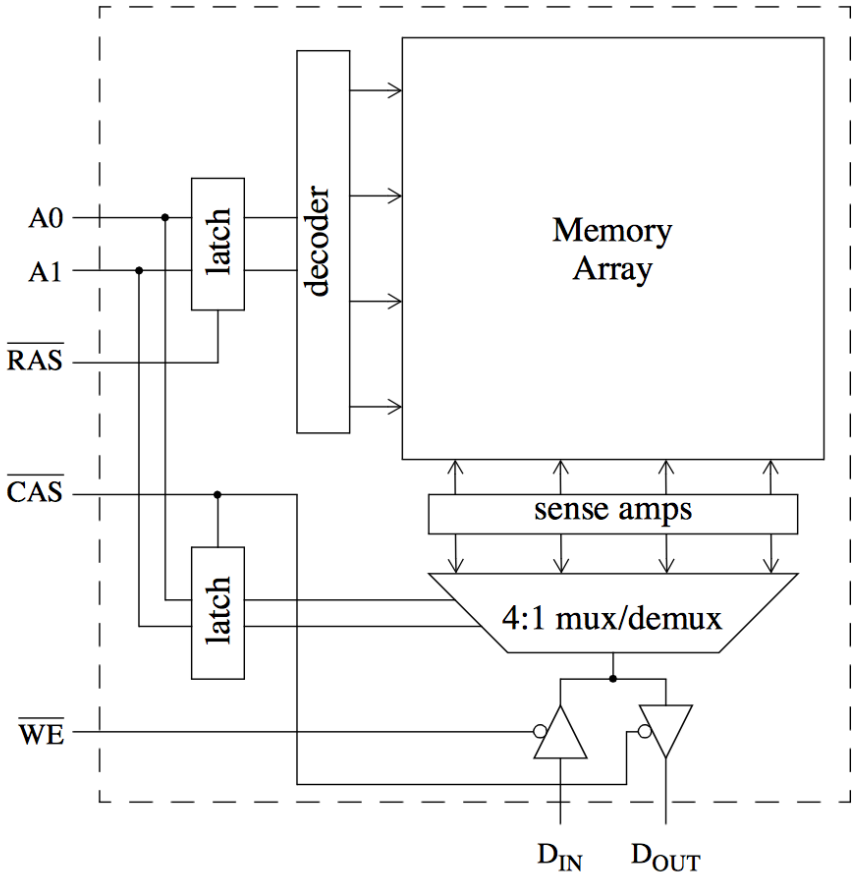


Figura 2.7 Diagrama de bloques de las memorias DRAM.

Las siguientes figuras 2.8 y 2.9 muestran los diagramas de tiempos para las operaciones de lectura y escritura respectivamente, en las memorias DRAM [20]:

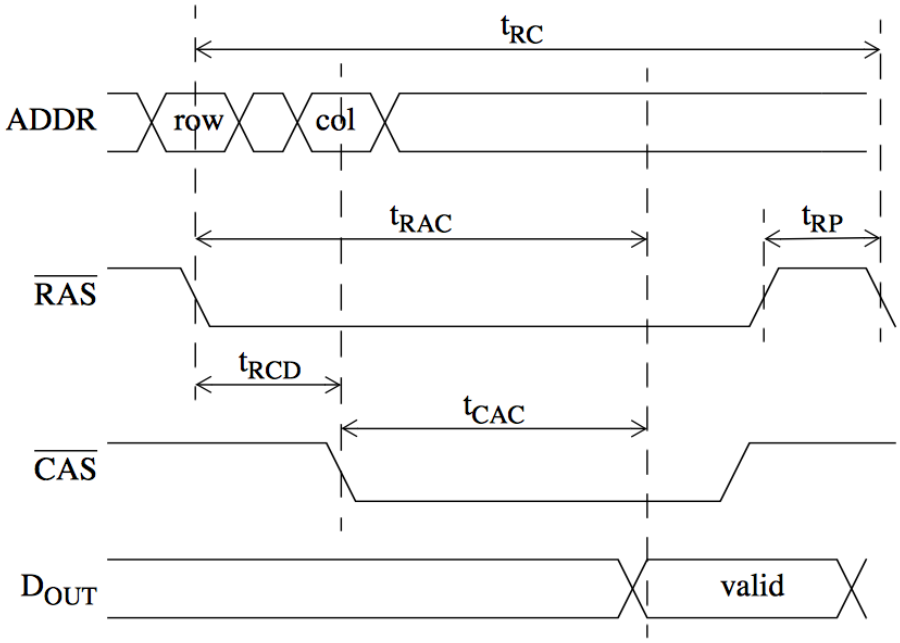


Figura 2.8 Diagrama de tiempos para la lectura de datos en memorias DRAM.

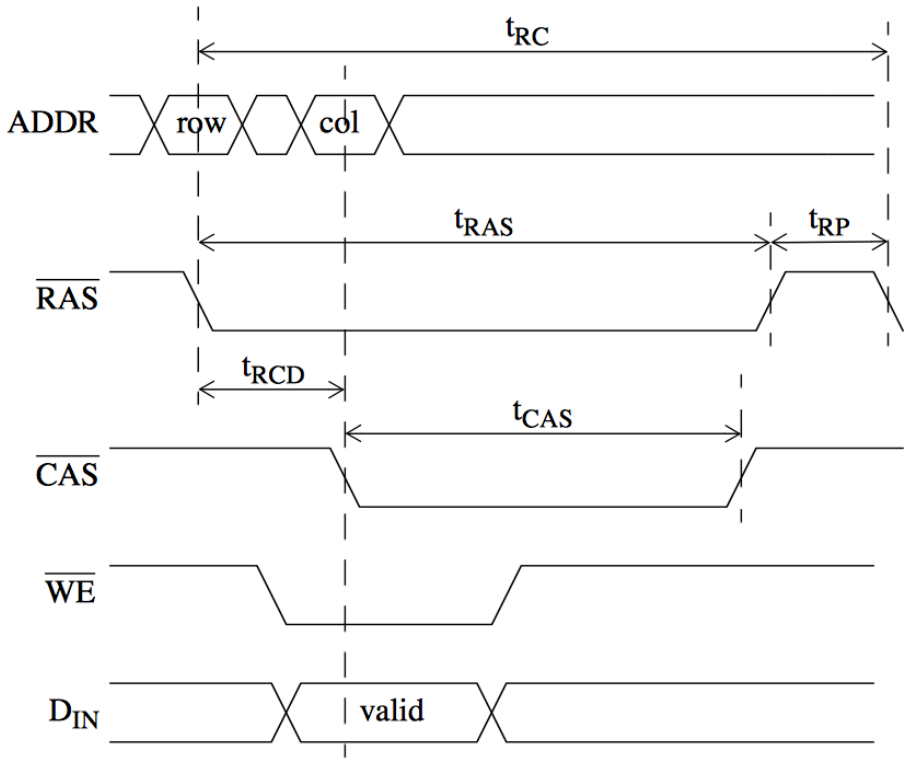


Figura 2.9 Diagrama de tiempos para la escritura de datos en memorias DRAM.

2.1.3.2 Memorias de acceso aleatorio no volátiles (ROM)

Las memorias ROM (“Read Only Memory”) son memorias que, a diferencia de las memorias RAM, no pierden la información si se interrumpe la alimentación eléctrica. Las aplicaciones más usuales son actuar como memoria de programa de un sistema basado en microprocesador o como almacenamiento de datos previo al apagado del sistema.

La interfaz de estas memorias es muy similar a la de las memorias RAM en cuanto la operación de lectura, variando en el proceso de escritura dependiendo de la tecnología con lo que ha sido fabricada. A continuación se detallan las tecnologías actuales para memorias ROM [21]:

- **No programable (ROM)**: En estas memorias los datos contenidos se deciden en el momento de la fabricación sin posibilidad de reescribir los datos.

- **Programable solo una vez (OTP-ROM)**: En las memorias OTP-PROM (“One Time Programmable Read Only Memory”) las celdas están constituidas por diodos. El contenido se almacena provocando una ruptura del diodo con una corriente excesiva, es decir, fundiéndolos.te

- **Borrables por ultravioletas (EPROM)**: Las memorias EPROM (“Eraseable Read Only Memory”) se basan en la tecnología MOS (“Metal-Oxide-Semiconductor”). Las celdas están formadas por transistores MOS especiales con una puerta flotante entre la puerta y el canal. Para borrar estas memorias es necesario radiarlas con rayos ultravioletas para que incidan en los electrones “atrapados” en las capas internas de los transistores, dándoles el potencial necesario para que se descarguen por la puerta flotante.

- **Borrables posición a posición (EEPROM)**: Las memorias EEPROM (“Electrically Eraseable Read Only Memory”) son la evolución de las EPROM. Permiten borrar el contenido eléctricamente sin necesidad de radiar con rayos ultravioleta.

- **Flash**: Las memorias flash son la evolución de las EEPROM, permitiendo mayor densidad de integración y tiempos de borrado inferiores. Además, se divide la matriz de celdas de memoria permitiendo borrar únicamente las secciones afectadas en la actualización de los datos, mejorando los tiempos de borrado.

2.2 Diseño e implementación del hardware del sistema propuesto

2.2.1 La placa de desarrollo DE2-115

2.2.1.1 Introducción

Este proyecto se desarrolla en la placa de desarrollo DE2-115 del fabricante Terasic (mostrada en la figura 2.10), la cual incorpora una FPGA de tipo Cyclone IVE del fabricante Altera. El dispositivo de captura de imagen utilizado es la cámara D5M de Terasic, compatible con la placa DE2-115. Para llevar a cabo el objetivo del proyecto se desarrolla un sistema hardware (tipo SoC) que integra los componentes necesarios para ejecutar una aplicación escrita en C/C++ para la detección de caras basada en el algoritmo de Viola-Jones.

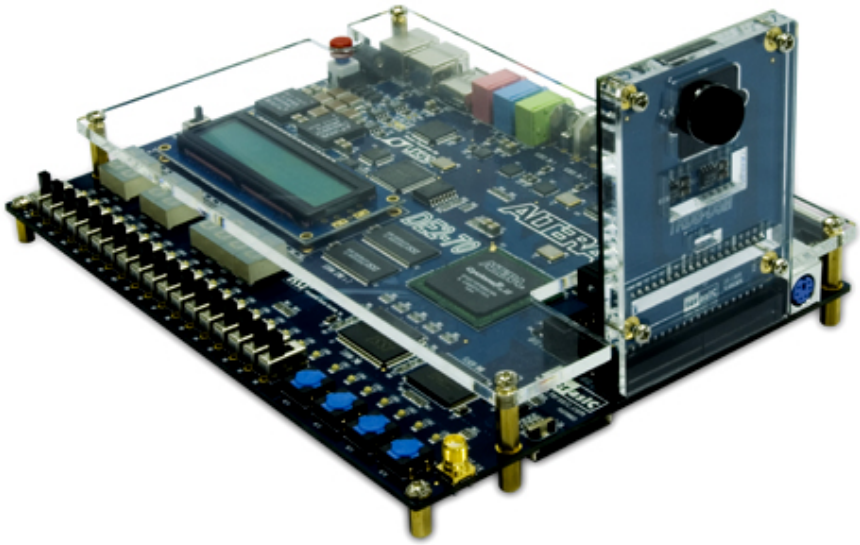


Figura 2.10 Placa de desarrollo DE2-115 con la cámara D5M acoplada.

La placa DE2-115, junto con la cámara D5M, posee todos los componentes necesarios para la realización de un sistema funcional de detección de caras. Esta placa incorpora entre sus interfaces una salida y dos entradas de audio tipo mini-jack, dos puertos USB que pueden actuar de esclavo o de maestro, una salida VGA, dos puertos Ethernet Gigabit, un puerto serie RS232 y un puerto serie PS/2 (utilizado comúnmente

para la entrada de un ratón o un teclado), entre otros. Estas interfaces nos proporcionan varias formas de control sobre el sistema a implementar. La figura 2.11 figura muestra un esquema de los distintos componentes e interfaces que incorpora la placa de desarrollo DE2-115.

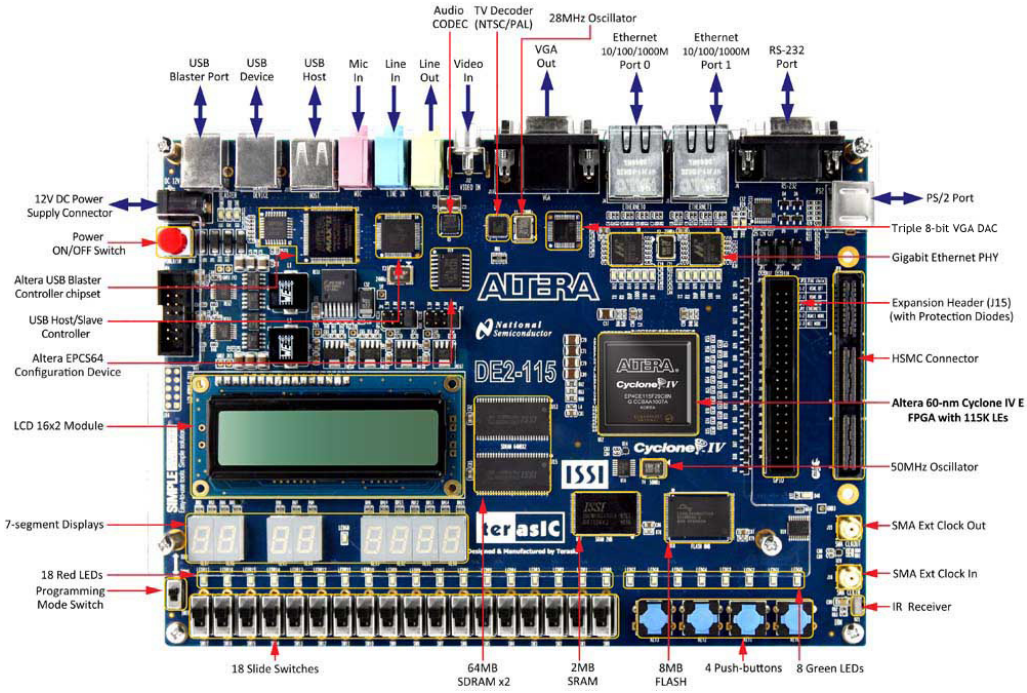


Figura 2.11 Esquema de componentes e interfaces de la placa de desarrollo DE2-115.

La figura 2.12 muestra las conexiones de los distintos componentes con la FPGA tipo Cyclone IV incorporada en la placa DE2-115.

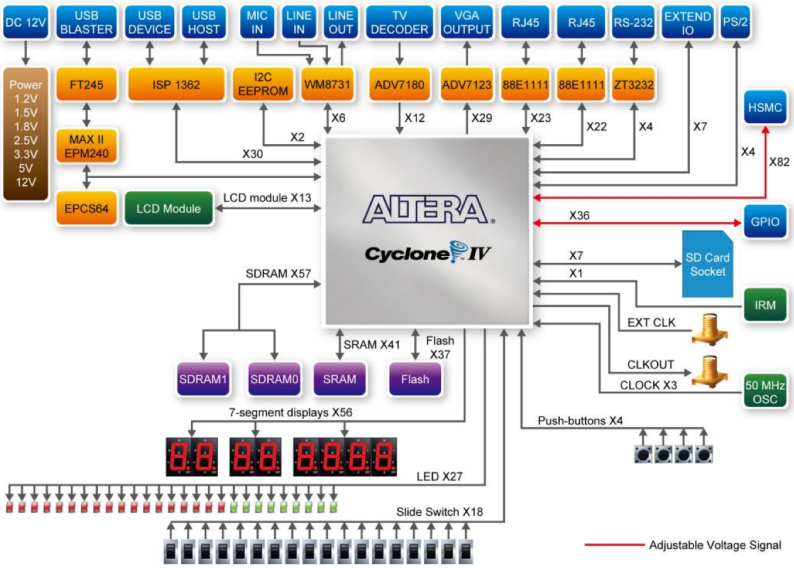


Figura 2.12 Conexiones entre componentes de la placa DE2-115.

En los siguientes apartados se detallan los componentes utilizados en este proyecto, donde se indica el número de pines de cada componente y una breve descripción, así como la dirección de los datos **(siempre referido a cada componente)** [22].

2.2.1.2 FPGA

La FPGA incluida en la placa DE2-115 es de tipo Cyclone IVE de Altera, modelo EP4CE115F29C7 el cual posee las siguientes características [23]:

- 114.480 elementos lógicos (LEs Logic Elements).
- 3.888 Kbits de memoria embebida.
- 266 multiplicadores embebidos de 9 bits.
- 4 periféricos PLLs.
- 528 pines de entrada y salida configurables por el usuario.

Los 114.480 elementos lógicos son suficientes para crear un sistema tipo SoC completo, incluso incorporando más de un procesador Nios II/f.

La memoria interna o embebida de la propia FPGA es la más limitada de la placa en cuanto a tamaño de almacenamiento, pero ofrece mayor velocidad de acceso con respecto a módulos de memoria externos debido a que carece de bus físico. Los 3.800 Kbits (aprox. 0.46 MB) de este tipo de memoria pueden parecer escasos para el almacenamiento de datos masivo, ya que su finalidad está orientada a ser usada como memoria cache del procesador o memoria FIFO para compensar cambios de reloj de un bus de datos.

Los multiplicadores embebidos de 9 bits aumentan el rendimiento reduciendo el coste computacional y el consumo de energía. Estos a su vez pueden ser combinados entre ellos para realizar operaciones a nivel de bit de 18x18. Los periféricos PLL permiten variar la frecuencia de un reloj digital por medio de la realimentación de la frecuencia y la fase. Los pines de entrada y salida de esta FPGA son 528, de los cuales 525 son utilizados para la conexión con los distintos componentes de la placa de desarrollo.

2.2.1.3 Memorias

La memoria DRAM (véase 2.3.1.2) de 128 MB incluida en la placa permite la creación de un sistema de tipo SoC bajo un sistema operativo basado en Linux como uClinux [24], donde es necesaria una capacidad de almacenamiento RAM alta. Por otro lado la placa DE2-115 también incluye una memoria FLASH (véase 2.3.2) de 8 MB que permite almacenar datos permanentemente, y una memoria SRAM (véase apartado 2.3.1.1) de 2 MB, siendo más rápida que la DRAM debido a que, en las memorias SRAM, las celdas de almacenamiento aíslan mejor los datos, aumentando el tamaño del integrado.

2.2.1.3.1 SDRAM

La memoria SDRAM se compone de **dos módulos** IS42S16320B del fabricante ISSI interconectados. Cada módulo es de 64 MB obteniendo una capacidad conjunta de 128 MB. El bus de datos es de 32 bits (16x2) y el bus de dirección de 13 bits. Básicamente para combinar los dos módulos en uno se utilizan las mismas señales de control para los dos módulos, incluido el bus de dirección. Por ejemplo si quisiéramos el dato de 32 bits de la dirección 10, el módulo 1 nos devolvería su dato de 16 bits de su dirección 10, y el módulo 2 su dato de 16 bits de su dirección 10, los cuales la FPGA recibe como la concatenación de ambos, es decir, un dato de 32 bits. La tabla 2.5 muestra los pines de los dos módulos utilizados conjuntamente.

Descripción	Número de pines	Dirección
Dirección (ADDR)	13	Entrada
Datos (DQ)	32	Entrada/Salida
Datos de máscara (DQM)	4	Entrada
Dirección de banco (BA)	2	Entrada
Selección de fila (RAS)	1	Entrada
Selección de columna (CAS)	1	Entrada
Habilitación de reloj (CKE)	1	Entrada
Reloj (CKE)	1	Entrada
Habilitación de escritura (WE)	1	Entrada
Habilitación de chip (CE)	1	Entrada

Tabla 2.5 Pines del módulo DRAM de la placa DE2-115.

NOTA: A partir de este apartado se tratarán los dos módulos físicos de memoria SDRAM como un único módulo, tal y como se describe en la tabla 2.5.

2.2.1.3.2 Flash

La memoria tipo FLASH es el modelo S29GL064N de 8MB del fabricante Spansion. Este tipo de memorias son capaces de almacenar datos permanentemente, es decir, aunque se corte el suministro eléctrico. Se pueden modificar los datos de la misma un número ilimitado de veces (sin contar deterioro por desgaste). El bus de datos de este módulo es de 8 bits y el bus de dirección de 23 bits. La tabla 2.6 muestra los pines del módulo.

Descripción	Número de pines	Dirección
Dirección (ADDR)	23	Entrada
Datos (DQ)	8	Entrada/Salida
Habilitación de lectura (OE)	1	Entrada
Habilitación de escritura (WE)	1	Entrada
Habilitación de selección de chip (CE)	1	Entrada
Habilitación de protección contra escritura (WP)	1	Entrada
Reseteo en alimentación, no datos (RST)	1	Entrada

Tabla 2.6 Pines del módulo Flash de la placa DE2-112.

2.2.1.3.3 SRAM

La memoria SRAM de la placa es una memoria asíncrona de 2MB modelo IS61WV102416BLL del fabricante ISSI. Su bus de datos es de un ancho de 16 bits y su bus de direcciones de 20 bits. La tabla 2.7 muestra los pines del módulo.

Descripción	Número de pines	Dirección
Dirección (ADDR)	20	Entrada
Datos (DQ)	16	Entrada/Salida
Habilitación de lectura (OE)	1	Entrada
Habilitación de escritura (WE)	1	Entrada

Habilitación de selección de chip (CE)	1	Entrada
Máscara byte bajo (LB)	1	Entrada
Máscara byte alto (UB)	1	Entrada

Tabla 2.7 Pines del módulo SRAM de la placa DE2-115.

2.2.1.4 Conversor digital-analógico VGA

El conversor digital-analógico, o DAC (Digital to Analog Converter) VGA, es un conversor de 24 bits que genera, a partir de señales digitales, las señales analógicas de la interfaz VGA. Los 24 bits corresponden a la señal RGB (Red-Green-Blue), siendo 8 bits para cada componente. A parte de los 24 bits que representan el color, posee las entradas/ salidas de control de la interfaz VGA. La tabla 2.8 muestra los pines del decodificador VGA.

Descripción	Número de pines	Dirección
Componente Rojo (VGA_R)	8	Entrada
Componente Verde (VGA_G)	8	Entrada
Componente Azul (VGA_B)	8	Entrada
Reloj (VGA_CLK)	1	Entrada
Señal blank (VGA_BLANK_N)	1	Entrada
Señal de sincronismo vertical (VGA_VS)	1	Entrada
Señal de sincronismo horizontal (VGA_HS)	1	Entrada
Señal de sincronismo (VGA_SYNC_N)	1	Entrada

Tabla 2.8 Pines del DAC VGA de la placa DE2-115.

2.2.1.5 Puerto GPIO

El puerto GPIO (General Purpose I/O) es un puerto de uso general que se compone de 40 pines, de los cuales 36 son configurables para entrada y salida y 4 para la alimentación eléctrica. Es utilizado en este proyecto para acoplar la cámara a la placa de desarrollo. La tabla 2.9 muestra los pines del módulo.

Descripción	Número de pines	Dirección
GPIO	36	Entrada/Salida

Tabla 2.9 Pines del puerto GPIO de la placa DE2-115.

2.2.1.6 LCD

La pantalla LCD permite la representación de 32 caracteres en dos líneas, es decir 16x2 caracteres. Este componente funciona con el controlador HD44780 ya integrado y puede generar todos los caracteres alfanuméricos. La tabla 2.10 muestra los pines del módulo LCD.

Dirección	Número de pines	Dirección
Datos (LCD_DATA)	8	Entrada/Salida
Escritura/Lectura (LCD_RW)	1	Entrada
Habilitación (LCD_EN)	1	Entrada
Selección de comando	1	Entrada
LCD encendido (LCD_ON)	1	Entrada
Luz de fondo (LCD_BLON)	1	Entrada

Tabla 2.10 Pines de la pantalla LCD de la placa DE2-115.

2.2.1.7 Displays de 7 segmentos

La placa dispone de 8 visualizadores de números hexadecimales de 7 segmentos (del pin 0 al 6). Esto hace un total de 56 pines para controlar los 8 visualizadores. La figura 2.13 muestra las conexiones entre un display de 7 segmentos y la FPGA.

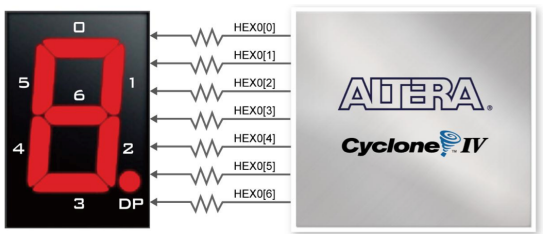


Figura 2.13 Conexiones entre un display de 7 segmentos con la FPGA Cyclone IV.

La tabla 2.11 muestra los pines de los visualizadores de 7 segmentos.

Descripción	Número de pines	Dirección
Display 0 (HEX0)	7	Entrada
Display 1 (HEX0)	7	Entrada
Display 2 (HEX0)	7	Entrada
Display 3 (HEX0)	7	Entrada
Display 4 (HEX0)	7	Entrada
Display 5 (HEX0)	7	Entrada
Display 6 (HEX0)	7	Entrada

Tabla 2.11 Pines de los displays de 7 segmentos de la placa DE2-115.

2.2.1.8 Pulsadores, interruptores y LEDs

La placa incluye 4 pulsadores, 18 interruptores, 18 LEDs (Light-Emitting Diode) rojos y 9 LEDs verdes. La tabla 2.12 muestra los pines de estos componentes.

Descripción	Número de pines	Dirección
Pulsadores (KEY)	4	Salida
Interruptores (SW)	18	Salida
LEDs rojos (LED_R)	18	Entrada
LEDs verdes (LED_G)	9	Entrada

Tabla 2.12 Pines de los pulsadores, interruptores y LEDs de la placa DE2-115.

2.2.1.9 Relojes

Esta placa incluye tres relojes que oscilan a 50 MHz. También se incluye una entrada para un reloj externo y una salida para exportar un reloj de la placa. Estos relojes son, respectivamente, CLOCK_50, CLOCK2_50, CLOCK3_50, SMA_CLKIN y SMA_CLKOUT. En este proyecto se utilizarán los dispositivos PLL integrados en la FPGA para modificar la frecuencia de los relojes a conveniencia de los componentes.

2.2.2 Implementación

La herramienta con la cual se realiza la implementación del sistema (SoC) es el entorno de desarrollo Quartus II de Altera; con ella se produce el análisis y la síntesis

para diseños escritos en HDL. Para este software se encuentran dos licencias, la “Edición Web” y la “Edición de Suscripción”, siendo la “Edición Web” con la que se realiza este proyecto al dar soporte a las FPGAs de bajo coste como son la familia Cyclone.

Este entorno permite al desarrollador crear los proyectos en lenguajes HDL (Verilog, VHDL, etc.), en un entorno gráfico para circuitos lógicos, o en una combinación de éstos. Por otro lado incluye “SOPC Builder”, un asistente para construir sistemas embebidos (SoC), el cual automatiza las conexiones de los componentes para crear un completo sistema compatible con cualquier FPGA de Altera. El asistente SOPC incluye una librería de componentes ya desarrollados, como por ejemplo el procesador Nios II, controladores de memoria, interfaces y periféricos, además de un asistente para componentes personalizados. El sistema creado para este proyecto contiene los componentes necesarios ejecutar el software de reconocimiento de caras. El componente principal será, aparte del procesador Nios II, el controlador de la cámara D5M de la cual recibiremos la imagen a analizar.

2.2.2.1 Módulo SoC

El módulo SoC que contiene el sistema hardware completo se crea con el programa SOPC Builder incluido en Quartus II. Los distintos componentes son interconectados mediante un bus. Cada componente opera con una frecuencia de reloj y con un ancho de datos especificado por el usuario; SOPC Builder automáticamente adaptará estas diferencias al bus de datos común. El sistema resultante devuelve los correspondientes códigos HDL de cada componente y un bloque gráfico, conjunto de todo ello, que se incorpora al esquema de Quartus II donde se interconectan las salidas y entradas de dicho bloque a los pines de la FPGA.

La figura 2.14 muestra el sistema SoC completo:

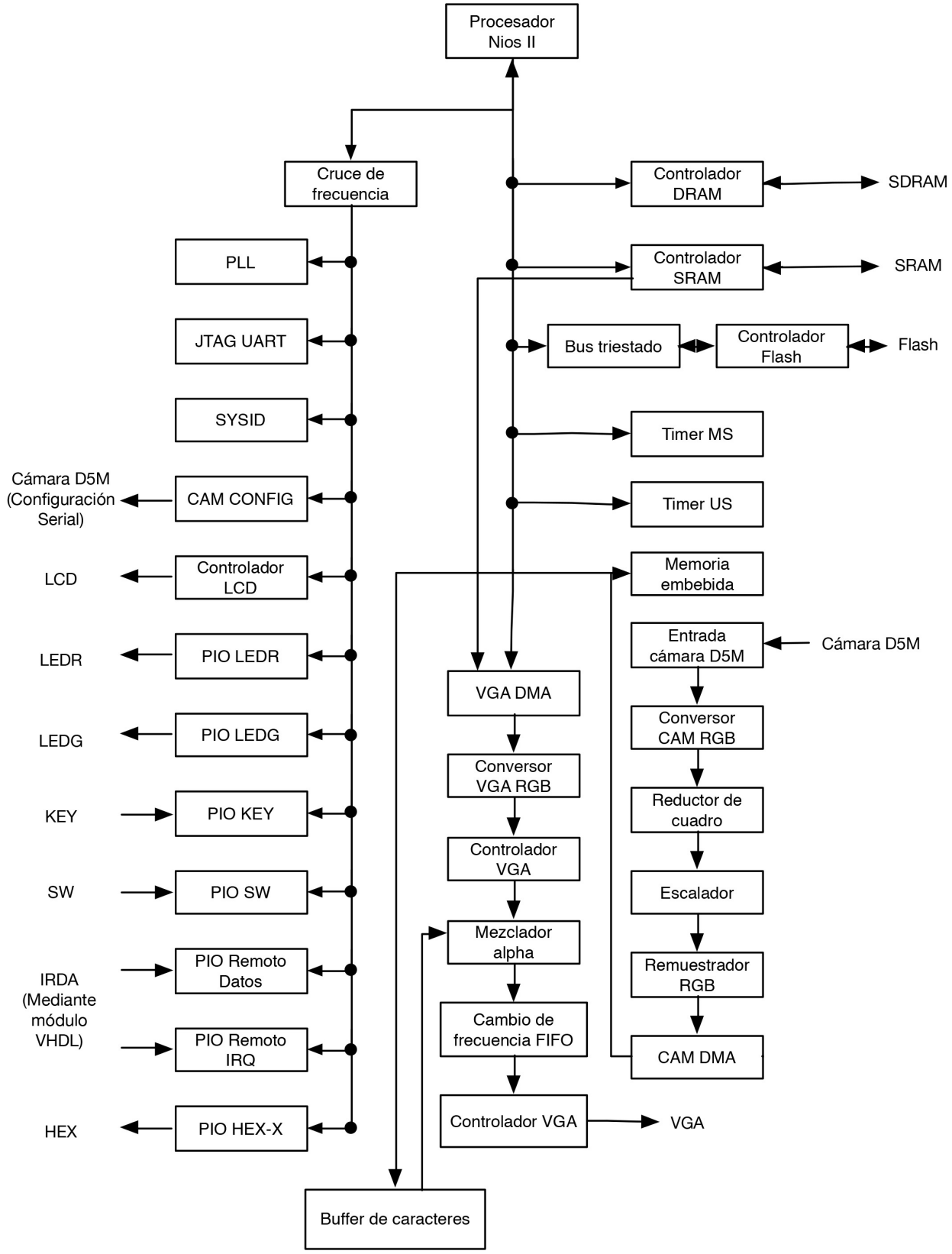


Figura 2.14 Esquema de conexiones del SoC.

2.2.2.1.1 Procesador Nios II/f

En este apartado utilizamos el procesador Nios II/f (denominado **CPU** en el sistema), el cual es un procesador de 32 bits, pudiendo manejar 32 interrupciones, y está específicamente diseñado para el alto rendimiento de aplicaciones con gran cantidad de datos y código, de hecho, este procesador puede albergar un sistema operativo Linux completo (uClinux). El problema de los procesadores embebidos es que no soporta altas velocidades de reloj como en el procesador de un PC, por ello la frecuencia máxima alcanzada en este sistema ha sido 85 MHz (**CLK_SYS**), ya que a frecuencias superiores se vuelve inestable. Los recursos utilizados por este procesador en términos de LEs son 1.800.

A continuación se muestran las características más relevantes del procesador Nios II/f que han sido utilizadas y/o configuradas:

- **Vector de excepciones**: Es la zona de memoria que almacena las direcciones de las excepciones del sistema. Es decir, si por ejemplo hay una división entre 0, provocaría un error que llamaría a una rutina alojada en la dirección que indique el vector de excepciones. En este proyecto este vector está alojado en la memoria SDRAM con un offset de 0x20.

- **Vector de reseteo**: Es la zona de memoria en la que el procesador se inicia como punto de partida. Esto no quiere decir que el programa este alojado en esa zona de memoria, sino que indica dónde va estar alojada la primera dirección del programa, la cual puede ser incluso en otro módulo de memoria. En este proyecto el vector de reseteo está alojado en la memoria SRAM con un offset de 0x0.

- **Memoria caché**: La memoria caché esta alojada en la memoria embebida del procesador y permite alojar los datos utilizados más frecuentemente disminuyendo el tiempo de acceso de las memorias externas. El procesador Nios II/f diferencia entre memoria caché de instrucciones (almacena las instrucciones más frecuentemente utilizadas) y la memoria caché de datos (almacena los datos más frecuentemente utilizados). Configuramos este tipo de memoria con un tamaño de 8 Kbytes para la memoria caché de datos y 16 Kbytes para la memoria caché de instrucciones.

- **Nivel de depuración**: El nivel de depuración permite establece las opciones de depuración en el entorno de desarrollo. Esta opción es configurada en nivel 1, que

permite la conexión mediante el componente JTAG, con puntos de interrupción por software. Este nivel de interrupción utiliza 400 LEs de los recursos de la FPGA.

- **Instrucciones personalizadas**: Un procesador posee un conjunto de instrucciones que incluyen la suma, la resta, la división, etc. Normalmente estas operaciones se realizan rápidamente para variables de números enteros, pero para variables de números decimales la operación supone un incremento del coste computacional importante, y por ello, aumentan los tiempos de ejecución. Si queremos operar con números decimales (variables de punto flotante) debemos incluir un módulo aparte específico para operaciones de punto flotante. En este caso se incluye una instrucción personalizada denominada “fpoint” que agiliza estas operaciones.

2.2.2.1.2 PLL

Los relojes que se utilizan son generados mediante un PLL (denominado **PLL** en el sistema) a partir de uno de los relojes de la placa de 50 MHz**,** denominado en el sistema SoC como **CLK_50**. La mayoría de los componentes trabajan a 50 MHz y, según las especificaciones del procesador Nios II, para este modelo de FPGA el límite está en 100 MHz aproximadamente, por ello se ha elegido 85 MHz para la CPU y las memorias, guardando un margen para evitar que se vuelva inestable. Por otro lado, es necesario desfasar el reloj de la memoria SDRAM con un retraso de 3-4 nanosegundos para contrarrestar los retardos del controlador DRAM/CPU y la memoria física en sí. La FPGA utilizada incorpora 4 PLLs, y cada PLL puede ser configurado para ofrecer 4 frecuencias diferentes, por lo que sólo es necesario utilizar un PLL con una frecuencia de entrada original de la propia placa (50 MHz). La tabla 2.13 muestra la configuración de los parámetros del PLL.

Nombre de salida	Factor de multiplicación	Factor de división	Desfase (ns)	Frecuencia de Salida (MHz)
CLK_SYS	17	10	0	85
CLK_SDRAM	17	10	-3,4	85
CLK_25	1	2	0	25

Tabla 2.13 Configuración del PLL.
(Frecuencia de entrada externa de 50 MHz)

Siendo:

- **CLK_SYS**: Reloj de 85 MHz utilizado para la CPU y las memorias SDRAM y FLASH.
- **CLK_SDRAM**: Reloj de 85 MHz desfasado -3,4 nanosegundos con respecto a CLK_SYS. El controlador SDRAM del sistema SoC trabaja con CLK_SYS, pero realmente la señal de reloj que enviaremos a la memoria física será ésta, CLK_SDRAM. Este retardo es necesario para contrarrestar los retardos de la CPU y la SDRAM, haciendo más estable la sincronización de reloj entre este componente y el resto del sistema.
- **CLK_25**: Reloj de 25 MHz utilizado para generar las señales VGA de 640x480 píxeles y para la sincronización de los datos de la cámara D5M .
- **CLK_50**: Este reloj lo proporciona la propia placa de desarrollo (denominada fuera del sistema SoC como CLOCK_50) y, aparte de utilizarlo para la señal de referencia del PLL, se usa para los componentes del SoC que requieren de una señal de 50 MHz explícitamente como se especifica en los siguientes apartados.

Datos generales:

- Conectado al reloj **CLK_50**, y a **CLOCK_CROSSING** (véase 3.2.1.7).
- Direcciones de memoria: **0x0b000040-0x0b00004f**.
- Sin interrupción.

2.2.2.1.3 Controlador SDRAM

Altera provee un controlador genérico configurable (denominado **SDRAM** en el sistema) para memorias SDRAM. Es necesario configurar este controlador con los parámetros adecuados de acuerdo con las especificaciones del módulo, como se muestra en la tabla 2.14.

Parámetro	Valor
Ancho de bus de datos	32 bits
Bancos de almacenamiento	4 bancos
Filas por banco	13 fil.
Columnas por banco	10 col.
Retardo de escritura/lectura (T_RCD)	20 ns
Tiempo de acceso (T_AC)	5,5 ns
Retardo de inicialización posterior a la alimentación	100 ns

Tiempo de precarga de los comandos	20 ns
Tiempo de refresco de los comandos	20 ns
Tiempo de recuperación de escritura	14 ns

Tabla 2.14 Configuración del controlador de la memoria SDRAM.

Datos generales:

- La frecuencia para este módulo es de 85 MHz sin retardo, es decir, **CLK_SYS**, pero una vez compilado el SoC, el módulo físicamente se conectará a **CLK_SDRAM** que conlleva un retardo de 3,4 nanosegundos. Sus salidas/entradas son dos, una hacia la memoria física SDRAM en el exterior del SoC, y otra conectada a los dos buses principales de instrucciones y datos del procesador.
- Direcciones de memoria: **0x00000000-0x07ffffff**.
- Sin interrupción.

2.2.2.1.4 Controlador Flash

La memoria flash de la placa es utilizada para almacenar los clasificadores para la detección de rostros y otros elementos gráficos de la interfaz final. El controlador (denominado **FLASH** en el sistema) de la memoria flash es "Flash Memory Interface (CFI)", el cual es configurado según se especifica en la tabla 2.15.

Parámetro	Valor
Ancho de bus de direcciones	23 bits
Ancho de bus de datos	8 bits
Tiempo de configuración	60 ns
Tiempo de espera	160 ns
Tiempo de mantenimiento de datos	60 ns

Tabla 2.15 Configuración del controlador de la memoria flash.

Este controlador tiene dos salidas, una externa que se conecta a la memoria física y otro al componente Nios II mediante un bus triestado. El bus triestado permite el valor 'Z' (impedancia), además de los valores binarios '1' y '0'. Es utilizado para interconectar el controlador de la memoria flash con los dos buses principales de instrucciones y datos del procesador NIOS II. El nombre original del componente es "Avalon-MM Tristate Bridge".

Datos generales:

- La velocidad de reloj a la que opera el bus es de 85 MHz (**CLK_SYS**) y es conectado a los buses principales de datos e instrucciones del procesador.
- Direcciones de memoria: **0x08800000-0x08ffffff**.
- Sin interrupción.

2.2.2.1.5 Controlador SRAM

El controlador "SRAM/SSRAM Controller" (denominado **SRAM** en el sistema) lo proporciona Terasic diseñado específicamente para la memoria SRAM de la placa DE2-115, por lo que no es necesario configurarlo.

Datos generales:

- Conectado a **CLK_50**. Sus salidas/entradas son dos, una conectada a la memoria física SRAM externa al SoC, y otra conectada a los dos buses de instrucciones y datos del procesador NIOS II.
- Direcciones de memoria:**0x09000000-0x091fffff**.
- Sin interrupción.

2.2.2.1.6 Memoria embebida

La memoria embebida (denominada **MEM_CAM** en el sistema) es la memoria donde se almacena la imagen captada por la cámara en un flujo constante. Esta memoria es embebida para maximizar el rendimiento. Su capacidad viene determinada por las dimensiones de la imagen a almacenar (320x240 pixeles) y su profundidad de color (16 bits), por lo que tenemos que:

320*240*16 = 1228800 bits = **153600 bytes** de memoria de cámara.

Datos generales:

- Esta memoria opera a 50 MHz (**CLK_50**) y es conectada al bus principal de datos del procesador y a **CAM_DMA** (véase 3.2.1.31).
- Direcciones de memoria: **0x08080000-0x08a57ff**.
- Sin interrupción.

2.2.2.1.7 Cruce de frecuencia reloj

El "Avalon-MM Clock Crossing Bridge" (denominado **CLOCK_CROSSING** en el sistema) permite interconectar los buses principales del procesador que operan a una

determinada frecuencia, con elementos de distinta frecuencia. Se podrían conectar estos elementos directamente al bus principal ya que SOPC Builder adaptaría estos cambios de reloj, pero no sería óptimo al bajar el bus principal de 85 MHz a los 50 MHz de otros elementos. Este módulo utiliza un buffer de memoria para adaptar las velocidades sin sacrificar la velocidad del bus principal.

Datos generales:

- Tiene dos puertos, uno de ellos opera a 85 MHZ (**CLK_SYS**) y se conecta a los dos buses principales de datos e instrucciones del procesador, y el otro puerto opera a 50 MHZ (**CLK_50**) y se conecta a los distintos componentes que operan a 50 MHz en el sistema.
- Direcciones de memoria: **0x0b000000-0b0001ff**.
- Sin interrupción.

2.2.2.1.8 Identificador del sistema

Este componente (denominado **SYSID**) dota al sistema de un identificador aleatorio para evitar que ejecutemos un programa desde Eclipse (parte de software) no compatible con este SoC.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y es interconectado al bus esclavo del puente de **CLOCK_CROSSING** (véase 3.2.1.7).
- Direcciones de memoria: **0x0b000180-0x0b0007**.
- Sin interrupción.

2.2.2.1.9 Comunicación serial

El módulo JTAG UART (denominado **JTAG_UART** en el sistema) implementa un método para comunicar mediante flujo en serie de caracteres el PC de desarrollo y el sistema SoC. Actúa como un puerto serie RS232, pero sin un puerto físico ya que los datos son mandados a través del USB con el que subimos la aplicación al sistema. Ha sido configurado con 64 bytes de buffer de entrada y 64 bytes de buffer de salida, además de habilitarse una señal de interrupción para este módulo con prioridad 0 (la más alta).

Datos generales:

- Este componente opera a 50 MHz (**CLK_50**) y es conectado al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7).
- Direcciones de memoria: **0x0b000020-0x0b000027**.
- Con interrupción de **prioridad 0**.

2.2.2.1.10 Configuración de la cámara

El nombre original del módulo es “Audio and Video ” (denominado **CAM_CONFIG** en el sistema) y es el encargado de controlar el módulo de la cámara D5M de Terasic, mediante datos serie.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y tiene dos puertos, uno conectado al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7), y otro conectado al módulo físico de la cámara D5M, concretamente a los pines SDAT y SCLK (mediante el puerto físico GPIO, véase 3.1.1.5) de la cámara en el exterior del SoC.
- Direcciones de memoria: **0x0b000000-0xb00000f**.
- Sin interrupción.

2.2.2.1.11 Controlador LCD

Este componente (denominado **LCD** en el sistema) permite controlar la pantalla de 16x2 caracteres de la placa mediante software.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y es conectado por un lado al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7), y por otro al módulo físico LCD de la placa, en el exterior del SoC.
- Direcciones de memoria: **0x0b000198-0x0b000199**.
- Sin interrupción.

2.2.2.1.12 Controlador LEDs rojos

Este componente (denominado **LEDR** en el sistema) permite el control de los 18 LEDs rojos en paralelo. Utiliza el controlador genérico “PIO (Parallel I/O)”.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y sus puertos se conectan, uno de ellos al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7) y el otro a los LEDs rojos (véase 3.1.1.8) de la placa de desarrollo.
- Direcciones de memoria: **0x0b0000a0-0x0b0000af**.
- Sin interrupción.

2.2.2.1.13 Controlador LEDs verdes

Este componente (denominado **LEDG** en el sistema) es idéntico al LEDR y permite el control de los 9 LEDs (véase 3.1.1.8) verdes en paralelo. Utiliza el controlador genérico "PIO (Parallel I/O)".

Datos generales:

- Opera a 50 MHz (**CLK_50**) y sus puertos se conectan, uno de ellos al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7), y el otro los LEDs verdes de la placa de desarrollo.
- Direcciones de memoria: **0x0b0000b0-0x0b0000bf**.
- Sin interrupción.

2.2.2.1.14 Controlador de pulsadores

Este componente denominado **KEY** en el sistema, se refiere a los 4 pulsadores de la placa; su controlador es "PIO (Parallel I/O)". A diferencia de LEDG y LEDR, los cuales son puertos de salida, el puerto de KEY es configurado como puerto de entrada. Solo utilizaremos en este apartado 3 pulsadores de los 4 que incluye la placa, reservándonos uno para la señal reset de nuestro sistema completo. Por otro lado se configura una interrupción para este dispositivo como flanco de bajada indistintamente del pulsador accionado.

Datos generales:

- Opera a una frecuencia de 50 MHz (**CLK_50**) y dos puertos se conectan al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7) y a los pulsadores físicos de la placa de desarrollo.
- Direcciones de memoria: **0x0b0000d0-0xb0000df**.
- Con interrupción de **prioridad 3 por flanco de bajada** (dado que los botones en reposo transmiten un 1 lógico, y al ser pulsados cambian a 0).

2.2.2.1.15 Controlador de interruptores

SW es el componente que controla los 18 interruptores de la placa. El controlador es "PIO (Parallel I/O)" configurado con 18 bits de entrada.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7) y a los interruptores físicos de la placa de desarrollo.

- Direcciones de memoria: **0x0b0000c0-0x0b0000cf**.
- Sin interrupción.

2.2.2.1.16 Controlador de datos del mando remoto

REMOTE_DATA es el componente por el cual se reciben los datos provenientes del dispositivo IRDA y a su vez del mando remoto. El controlador es ”PIO (Parallel I/O)” configurado para 16 bits de entrada.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7) y al módulo externo al SoC que gestiona la entrada de datos del puerto IRDA por el puerto DATA.
- Direcciones de memoria: **0x0b0000e0-0x0b0000ef**.
- Sin interrupción.

2.2.2.1.17 Controlador de IRQ del mando remoto

REMOTE_IRQ es el componente por el cual se recibe la interrupción que indica que hay un nuevo dato de entrada. El módulo de control remoto externo (véase 3.2.2) pasa de 0 a 1 y se mantiene un ciclo de reloj para volver a 0 posteriormente, esta señal es captada como una interrupción por flanco de subida para especificar que existe una nueva entrada de datos. El controlador es ”PIO (Parallel I/O)” configurado para 1 bit de entrada.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7) y al módulo externo al SoC que gestiona la entrada de IRDA, por el puerto IRQ.
- Direcciones de memoria: **0x0b000010-0x0b00001f**.
- Con interrupción de **prioridad 4 por flanco de subida**.

2.2.2.1.18 Controladores de displays hexadecimales

Los HEX (denominados **HEX0 - HEX7** en el sistema) hacen referencia a los 8 displays hexadecimales, por ello creamos 8 componentes: HEX0, HEX1, HEX2 HEX3, HEX4, HEX5, HEX6 y HEX7.

El controlador de cada uno es “PIO (Parallel I/O)” configurados con 4 salidas. Cada display físico tiene 7 entradas correspondientes a cada segmento del display, pero si queremos que represente un número hexadecimal solo hacen falta 4 bits (2^4=16).

Datos generales:

- Cada componente opera a 50 MHz (**CLK_50**) y se conecta por un lado al bus esclavo del puente **CLOCK_CROSSING** (véase 3.2.1.7), y por otro lado a los decodificadores 7447 (véase 3.2.3). Los decodificadores 7447 convertirán estos datos hexadecimales de 4 bits en datos de 7 bits para poder ser representados en los displays.
- Direcciones de memoria: **0x0b000100-0x0b00017f**.
- Sin interrupción.

2.2.2.1.19 Temporizadores

Los timers o temporizadores (denominados **TIMER_MS** y **TIMER_US** en el sistema) se emplean para controlar periodos de tiempo. Básicamente son contadores que expresan el tiempo transcurrido desde que se inició hasta el momento de la lectura. En este proyecto se han creado dos timers, uno cuenta en milisegundos (TIMER_MS) y el otro en microsegundos (TIMER_US). Además ha sido habilitada la señal de interrupción para cada timer.

Datos generales:

- Los timers operan a 85 MHz (**CLK_SYS**) y son conectados al bus principal de datos.
- Direcciones de memoria: **0x08010000-0x0801003f**.
- Con interrupción de **prioridad 2 para TIMER_MS y prioridad 1 para TIMER_US**.

2.2.2.1.20 Controlador de acceso a memoria VGA

El componente VGA_DMA (denominado **VGA_DMA** en el sistema) permite el acceso directo a la memoria SRAM donde se almacenan dos buffers del tamaño de la imagen a mostrar, lo que libera al procesador de la tarea de atender el acceso. Los datos de la memoria son recibidos en modo *streaming* y devueltos al componente VGA_RGB.

640x480x16x2buffers = 9.830.400 bits = 1.228.800 bytes = 1200 KBytes.

Datos generales:

- Este componente opera a 50 MHz (**CLK_50**) y se conecta al controlador SRAM, al bus principal de datos y al módulo **VGA_RGB** (véase 3.2.1.21).
- Direcciones de memoria: **0x08042000-0x0804200f**.
- Sin interrupción.

2.2.2.1.21 Conversor RGB del VGA

Este componente (denominado **VGA_RGB** en el sistema) es el encargado de convertir la profundidad de los pixeles en 16 bits a 30 bits. Esto es necesario porque el resto de elementos de la cadena VGA trabajan una profundidad de pixel de 30 bits.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta por un lado a **VGA_DMA** (véase 3.2.1.20) y por otro lado al módulo **VGA_ALPHA** (véase 3.2.1.23).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.22 Búffer de caracteres

Este componente (denominado **VGA_CHAR_BUFFER** en el sistema) almacena una librería de caracteres para ser representados mediante la interfaz VGA y nos libera de la labor de crear una imagen por píxeles de cada caracter.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta por un lado al bus principal del procesador y por otro lado a **VGA_ALPHA** (3.2.1.21).
- Direcciones de memoria: **0x08042020-0x08042027**.
- Sin interrupción.

2.2.2.1.23 Mezclador Alpha

Este componente (denominado **VGA_ALPHA** en el sistema) fusiona las salidas de VGA_CHAR_BUFFER y VGA_RGB. Esto quiere decir que la salida de VGA_RGB se visualizará por debajo de la salida VGA_CHAR_BUFFER. Dicho de otro modo, se representarán los caracteres que nosotros queramos y la imagen almacenada en la SRAM detrás de estos.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta a las salidas de **VGA_CHAR_BUFFER** (véase 3.2.1.22) y **VGA_RGB** (véase 3.2.1.21), y a la entrada de **VGA_CLOCK** (véase 3.2.1.24).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.24 Cambio de frecuencia FIFO

Este componente (denominado **VGA_CLOCK** en el sistema) convierte el dominio de reloj de los datos en streaming (devueltos por "Mezclador alpha"), de 50 MHz a 25 MHz. Esto es necesario porque el controlador VGA que genera las señales trabaja a 25 MHz.

Datos generales:

- Opera con dos relojes, 50 MHz y 25 MHz (**CLK_50** y **CLK_25**), para su respectiva entrada y salida de datos. Sus conexiones son a **VGA_ALPHA** (véase 3.2.1.23) y a **VGA_CONTROLLER** (véase 3.2.1.25).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.25 Controlador VGA

Este componente (denominado **VGA_CONTROLLER** en el sistema) genera las señales de sincronismo de la interfaz VGA y los devuelve al exterior del SoC junto a los datos RGB de la imagen. A pesar de recibir los datos de la imagen con una profundidad de 30 bits, la salida que proporciona fuera del sistema es de 24 bits, por lo que ya están adaptados al conversor digital-analógico VGA de la placa. Las señales que proporciona este controlador son las establecidas para dar salida a una imagen de 640x480 pixeles, y por lo tanto es necesario un reloj de frecuencia 25 MHz como se muestra en la tabla 2.16.

Parametro	Valor
Frecuencia de reloj	**25,175 MHz**
Tasa de refresco	**60 Hz**
Área visible vertical	480 líneas
Porche frontal vertical	2 líneas

Porche posterior vertical	25 líneas
Pulsos de sincronismo vertical	2 líneas
Borde izquierdo vertical	8 líneas
Borde derecho vertical	8 líneas
Total lineas verticales	**525 líneas**
Área visible horizontal	640 líneas
Porche frontal horizontal	8 líneas
Porche posterior horizontal	40 líneas
Pulsos de sincronismo horizontal	96 líneas
Borde izquierdo horizontal	8 líneas
Borde derecho horizontal	8 líneas
Total lineas horizontales	**800 líneas**

Tabla 2.17 Tabla de sincronismo VGA para 640x480 pixeles.

Por lo que tenemos que:

800x525x60Hz = 25,2 MHz de frecuencia de reloj necesaria

La emisión de la imagen se produce mediante un barrido realizado línea por línea de arriba abajo hasta completar las 480 líneas verticales. Por cada línea vertical, a su vez, despliega 640 píxeles que conforman la imagen. La figura 2.15 muestra el barrido realizado en el cuadro de la imagen.

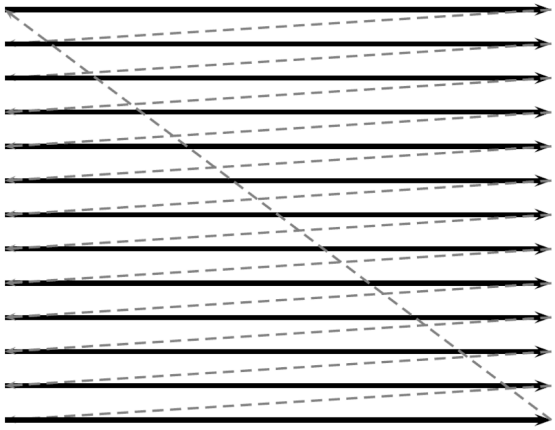


Figura 2.15 Modo barrido en el formato VGA.

Este barrido es visible si se escribe en el mismo buffer que se está mostrando, por lo que para eliminar este artefacto es necesario un segundo buffer que nos permita intercambiar entre ambos buffers sin necesidad de escribir en el buffer mostrado en ese momento (véase 3.3.3.2).

Datos generales:

- Opera a 25 MHz (**CLK_25**) y se conecta a **VGA_CLOCK** (véase 3.2.1.24) y a los pines del DAC VGA de la placa de desarrollo.
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.26 Entrada Cámara D5M

Este componente (denominado **CAM_IN** en el sistema) captura los datos en modo streaming de la cámara D5M para integrarlos en el SoC. La imagen que proporciona la cámara D5M es de una resolución de 2592x1944 con una profundidad de 8 bits en modo Bayer Pattern (RAW).

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta a la cámara mediante el puerto físico GPIO de la placa de desarrollo y a l componente **CAM_RESAMPLER** (véase 3.2.1.27).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.27 Remuestrador RGB

Este componente (denominado **CAM_RESAMPLER** en el sistema) convierte la imagen devuelta por la cámara, que tiene un formato Bayer Pattern (RAW) a un formato RGB (en el que ya se distingue 3 componentes). En la estructura RAW cada pixel solo es un componente del RGB. Lo que vemos es que para convertir a un pixel RGB es necesario 4 (array de 2x2) pixeles del formato RAW, por ello la resolución pasa de 2592x1944 pixeles (RAW) a 1296x972 pixeles (RGB 24 bits). El porqué de necesitar 4 componentes RAW para los 3 componentes RGB es porque en RAW la información del componente verde es de 16 bits (8 bits x 2 píxeles), mejorando la respuesta del ojo humano ya que es más sensible a este color.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta a **CAM_IN** (véase 3.2.1.26) y a CAM_CLIPPER (véase 3.2.1.28).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.28 Reductor de cuadro

Este componente (denominado **CAM_CLIPPER** en el sistema) se utiliza para recortar la imagen devuelta por CAM_RESAMPLER de una resolución de 1296x972 pixeles a 1280x960 pixeles. Es necesario debido a que CAM_SCALER solo escala a una proporción de 0.25, 0.5, 2 y 4. Estas proporciones permitirían escalar desde 1296x972 a 160x120 píxeles, que es a la que se trabaja con la imagen de la cámara en este proyecto.

Datos generales:

- Opera a 50 MHz (**CLK_C50**) y es conectado a **CAM_RESAMPLER** (véase 3.2.1.27) y a **CAM_SCALER** (véase 3.2.1.29).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.29 Escalador

Este componente (denominado **CAM_SCALER** en el sistema) escala la imagen entrante de 1280x960 píxeles a 320x240 píxeles mediante un factor de 0.25. Es necesario escalarlo por falta de almacenamiento en la memoria embebida. El tamaño de la imagen resultante viene dado por:

Eje x: 1280 x 0.25 = 160 píxeles
Eje y: 960 x 0.25 = 120 píxeles

Datos generales:

- Los dos componentes operan a 50 MHz (**CLK_50**) y son conectados a **CAM_CLIPPER** (véase 3.2.1.28) y a **CAM_RGB** (véase 3.2.1.30).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.30 Conversor RGB de la cámara

Este componente (denominado **CAM_RGB** en el sistema) convierte la profundidad de color de 24 bits por pixel a 16 bits por pixel. Es necesario disminuirlo por falta de almacenamiento.

- 320*240 = 76.800 pixeles/imagen -> 76.800*24 = 1.843.200 bits por imagen de 24 bits.
- 320*240 = 76.800 pixeles/imagen -> 76.800*16 = 1.228.800 bits por imagen de 16 bits.

Por lo que tenemos un ahorro de memoria de un 33 % respecto al tamaño de la imagen de 24 bits.

Datos generales:

- Este componente opera a 50 MHz (**CLK_50**) y se conecta a **CAM_SCALER** (véase 3.2.1.29) y a **CAM_DMA** (véase 3.2.1.31).
- Sin direcciones de memoria.
- Sin interrupción.

2.2.2.1.31 Controlador de acceso a memoria CAM

Este componente (denominado **CAM_DMA** en el sistema) es el puente que permite almacenar los datos en streaming de la cámara en la memoria MEM_CAM.

Datos generales:

- Opera a 50 MHz (**CLK_50**) y se conecta a **CAM_RGB** (véase 3.2.1.30) y a **MEM_CAM** (véase 3.2.1.6).
- Direcciones de memoria: **0x08000000-0x0800000f**.
- Sin interrupción.

2.2.2.2 Módulo de control remoto

El módulo para el control remoto por infrarrojos no está incluido en el SoC, y ha sido escrito en lenguaje VHDL. Este componente tiene dos tipos de salida, una es un bus 32 bits, de los cuales 16 representan la tecla, y un bit que indica si se ha pulsado una tecla nueva. Este componente es proporcionado por Terasic.

2.2.2.3 Módulos 7447

Los módulos 7447 adapta un número representado de forma binaria a los LEDs correspondientes de los displays de 7 segmentos para representarlos en forma decimal. Son proporcionados por Altera.

2.2.2.4 Implementación final de hardware

En este apartado se explica la implementación del módulo SoC, los módulos 7447 y el modulo para el control remoto, con la FPGA Cyclone IV E incorporada en la placa DE2-115. Una vez compilado e implementado el sistema total en la FPGA, se puede ejecutar cualquier programa que hayamos escrito en C/C++ o ensamblador.

El módulo SoC creado se visualiza como un bloque con las entradas y salidas para ser conectados a los pines I/O de la FPGA, los cuales han sido renombrados para facilitar su función (por ejemplo CLOCK_50 es el PIN_Y2 físico de la FPGA). El esquema resultante del sistema con todos los módulos interconectados se encuentra en el Anexo I.

Tras la compilación de todo el sistema, los recursos libres y utilizados se muestran en la tabla 2.18.

Recurso	Uso
Elementos lógicos	10.700/114.80 (9%)
Registros totales	7302
Pines	317/529 (60%)
Memoria embebida (bits)	1.676.003/3.981.312 (42%)
Multiplicadores embebidos de 9 bits	11/532 (2%)
PLLs	1/4 (25%)

Tabla 2.18 Recursos hardware utilizados para la creación del sistema.

La figura 2.16 muestra el sistema completo (hardware) con el SoC, el módulo para la lectura del mando remoto y los módulos decodificadores 7447.

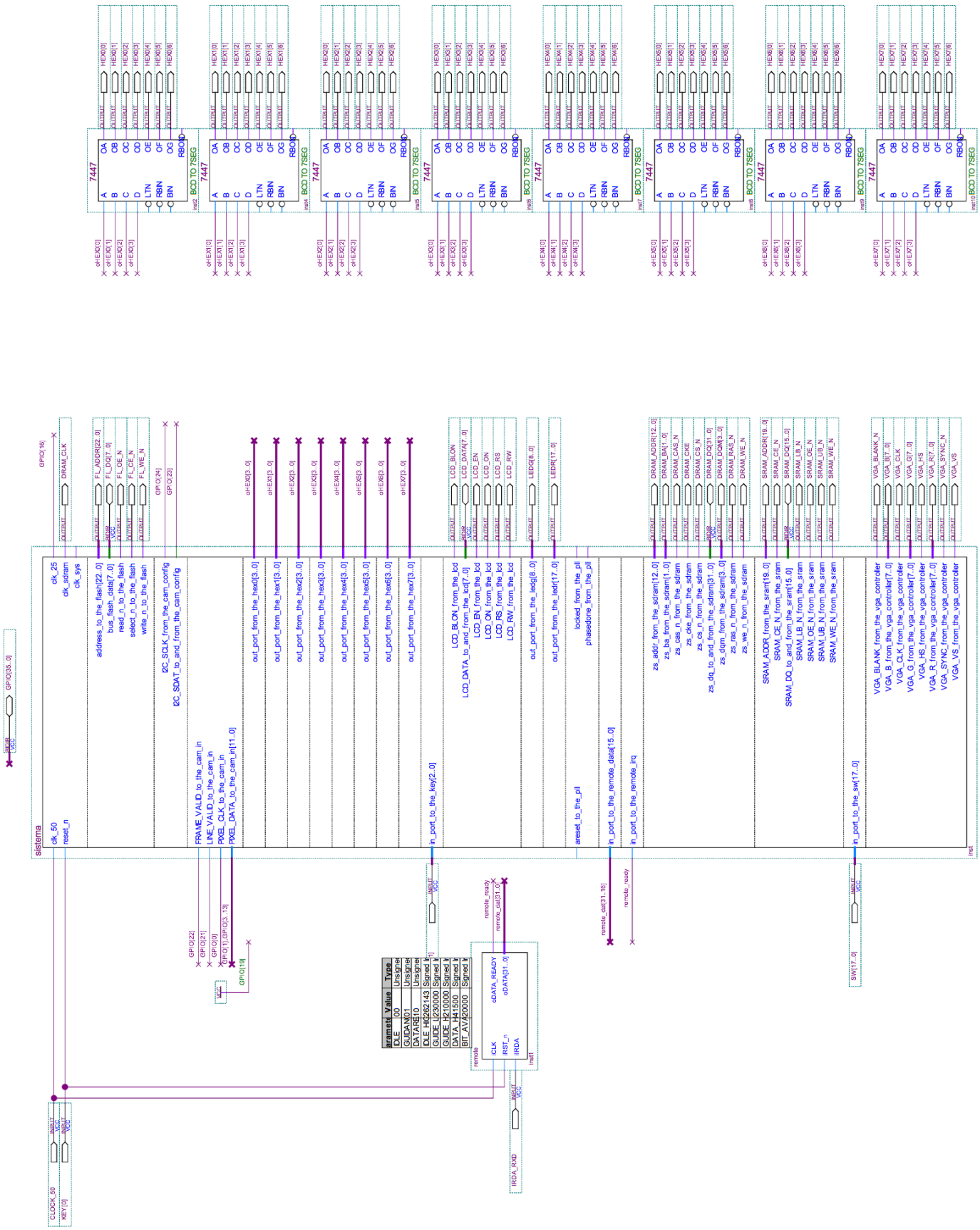


Figura 2.16 Esquema final del sistema (hardware).

Capítulo 3

Implementación del algoritmo de deteccion de caras

3.1 Estado del Arte

3.1.1 Algoritmo de Viola-Jones

El algoritmo de Viola-Jones (VJ) es un algoritmo de detección de caras con un coste computacional bajo propuesto por Paul Viola, de Mitsubishi Electric Research Labs, y Michael Jones, de Compaq CRL, en julio de 2001 [25].

El trabajo de Viola y Jones realiza tres contribuciones fundamentales: la creación de una nueva representación de la imagen que disminuye los tiempos de extracción de las características (imagen integral); la construcción de un clasificador basado en AdaBoost; y la combinación de clasificadores en cascada.

3.1.1.1 Imagen integral

La primera contribución es una nueva representación de la imagen denominada *imagen integral* que permite una rápida extracción de características al disminuir considerablemente el número de operaciones sobre los píxeles. El tamaño de la imagen integral es el mismo que el de la imagen original y **sus píxeles contienen la información de luminancia acumulada** en la región entre el punto de coordenadas (0, 0) y el pixel en cuestión.

La Figura 3.1 muestra la región de luminancia acumulada por un punto de la imagen integral (x,y).

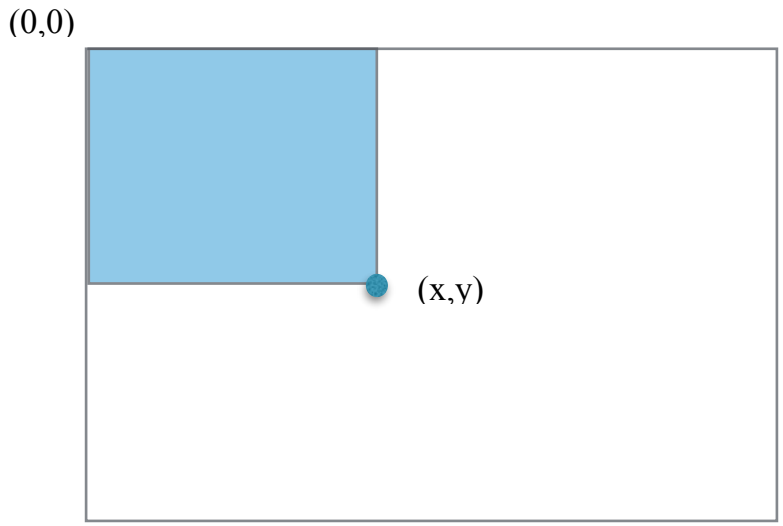


Figura 3.1 La imagen integral. El valor del punto (x, y) de la imagen integral es la suma de la luminancia de todos los píxeles contenidos en la región superior izquierda a este punto.

Siendo *ii*(x, y) la imagen integral e *i*(x, y) la luminancia de la imagen original tenemos que:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

Lo que puede calcularse a partir de las siguientes ecuaciones recurrentes:

$$s(x, y) = s(x, y - 1) + i(x, y)$$
$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

donde *s*(x ,y) es la acumulación de la fila, teniendo en cuenta que *s*(x, -1)=0 y que *ii(*-1,y)=0. Gracias a la imagen integral podemos obtener las luminancias acumuladas en las regiones a analizar.

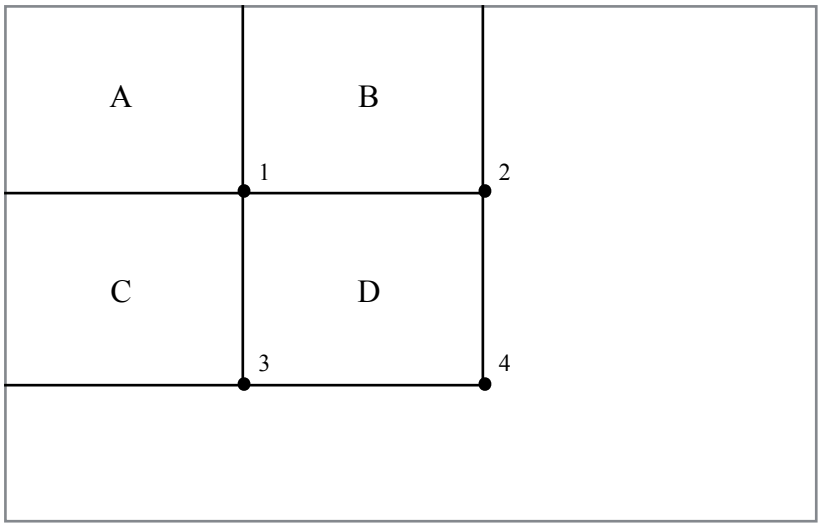


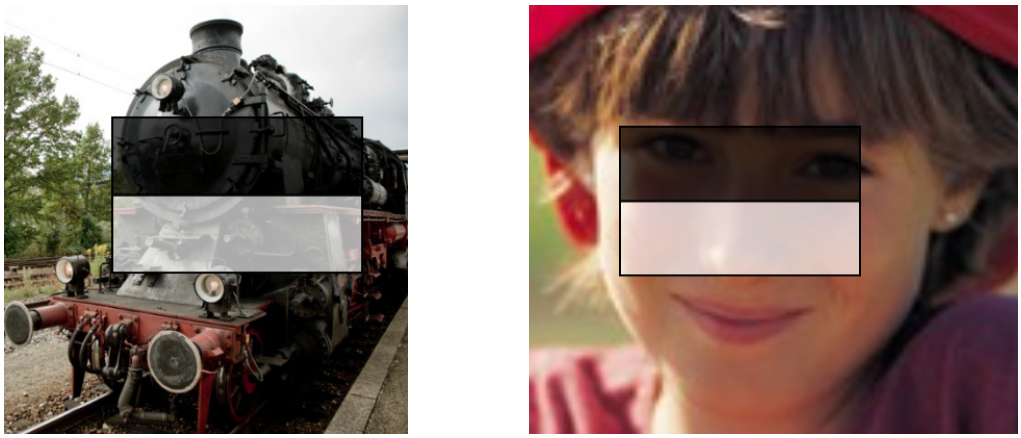
Figura 3.2 Subregiones en la imagen integral.

Por ejemplo, teniendo en cuenta las regiones mostradas en la figura 3.2, la suma de la luminancia de los píxeles incluidos en la región D puede ser hallada con cuatro puntos de la imagen integral. El valor del punto 1 es la acumulación de luminancia de la región A, el valor del punto 2 es la acumulación de la región A+B, el punto 3 es la suma de A+C y el punto 4 es la suma de A+B+C+D. Por lo que para hallar la suma correspondiente a la región D tenemos que:

$$D = 4 + 1 - 2 - 3$$

Con este procedimiento podemos hallar los valores necesarios para evaluar las características requeridas por el clasificador.

En particular, el algoritmo trabaja sobre un conjunto de características sencillas que se pueden extraer muy rápidamente a diferentes escalas a partir de la imagen integral. Estas características evalúan si existen determinados rasgos de la cara en la zona de detección, como se muestra en la figura 3.3.



a) b)

Figura 3.3 Ejemplo de característica para evaluar la existencia del conjunto de ojos y nariz. a) Evaluación sobre un ventana que no es una cara; b) Evaluación sobre una ventana con una cara. [26].

Las características se componen de rectángulos que abarcan un conjunto de píxeles cuya suma de niveles de gris (luminancia) se utiliza para la evaluación. En este ejemplo la diferencia entre la zona negra y la blanca devuelve un valor que se comparará con un umbral determinado en el clasificador para evaluar la existencia del rasgo.

Valor de la característica = ∑(Rectángulo blanco)-∑(Rectángulo sombreado)

Las características básicas contienen dos rectángulos, pero a medida que se superan las etapas se utilizan características más complejas formadas por tres o cuatro rectángulos. La figura 3.4 muestra tres ejemplos de estas características.

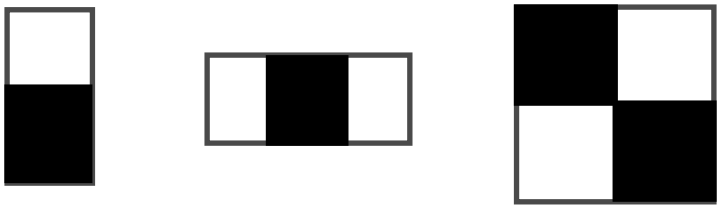


Figura 3.4 Ejemplos de características utilizadas por el clasificador.

Uno de los primeros pasos del clasificador es comprobar si pueden existir ojos y nariz, por ello la característica en cuestión constará de tres rectángulos: uno sobre el ojo izquierdo, otro sobre la nariz y otro sobre el ojo derecho. La diferencia entre los valores de luminancia proporcionados por los rectángulos de los ojos y el rectángulo de la nariz resultarán en un valor a partir del cual se determinará si existe la posibilidad de que haya

una nariz y ojos, o no. La figura 3.5 muestra de manera representativa la característica sobre la ventana de la imagen.

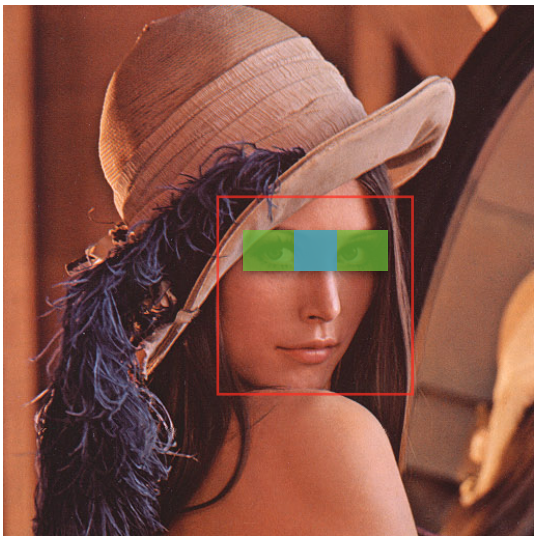


Figura 3.5 Representación de una característica sobre la imagen.

En la figura 3.5 el cuadro de borde rojo indica la ventana a procesar, los recuadros verdes indican que su valor de luminancia será sumado con signo positivo y el recuadro azul con signo negativo, para hallar el valor final a comparar con el umbral del clasificador.

3.1.1.2 Selección de características mediante boosting

La segunda contribución es un método para construir un clasificador mediante la selección de un pequeño número de características importantes mediante Adaboost [27]. Dentro de cada sub-imagen objeto de estudio el número de total de posibles características es muy grande, mucho más que el número de píxeles. Para garantizar una clasificación rápida, el proceso de aprendizaje debe excluir una gran mayoría de las posibles características, y centrarse en un pequeño conjunto de características críticas. En el algoritmo propuesto, se definen clasificadores débiles (sencillos y poco precisos) que se apoyan en una única característica. Como resultado, cada etapa del proceso de "boosting", que selecciona un nuevo clasificador débil, puede verse como un proceso de selección de características. Es necesario un entrenamiento de los clasificadores con caras en diferentes posiciones y de diferentes tamaños. En la figura 3.6 se muestran dos características obtenidas después del entrenamiento con una probabilidad de detección del 100%, pero con una probabilidad de falsa alarma del 50%, la cual no resulta admisible.

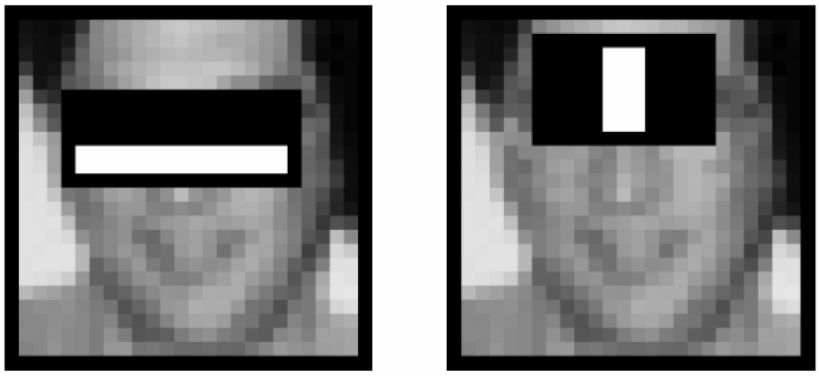


Figura 3.6 Características entrenadas con AdaBoost.

El diseño de las características se realiza en ventanas de 20x20 píxeles, siendo posible escalarlas a otros tamaños para la detección de caras mas alejadas o mas cercanas en la imagen.

3.1.1.3 Cascada de clasificadores para rechazo rápido

La tercera contribución del trabajo es un método para combinar clasificadores sucesivamente más complejos en cascada, lo que incrementa dramáticamente la velocidad del detector centrando la atención en las ventanas prometedoras de la imagen. La idea es determinar rápidamente dónde puede aparecer una cara en una imagen y reservar el procesado más complejo para estas ventanas. De esta forma, la probabilidad de falsa alarma disminuye y la probabilidad de detección aumenta a medida que se superan las etapas. La figura 3.7 ilustra este proceso.

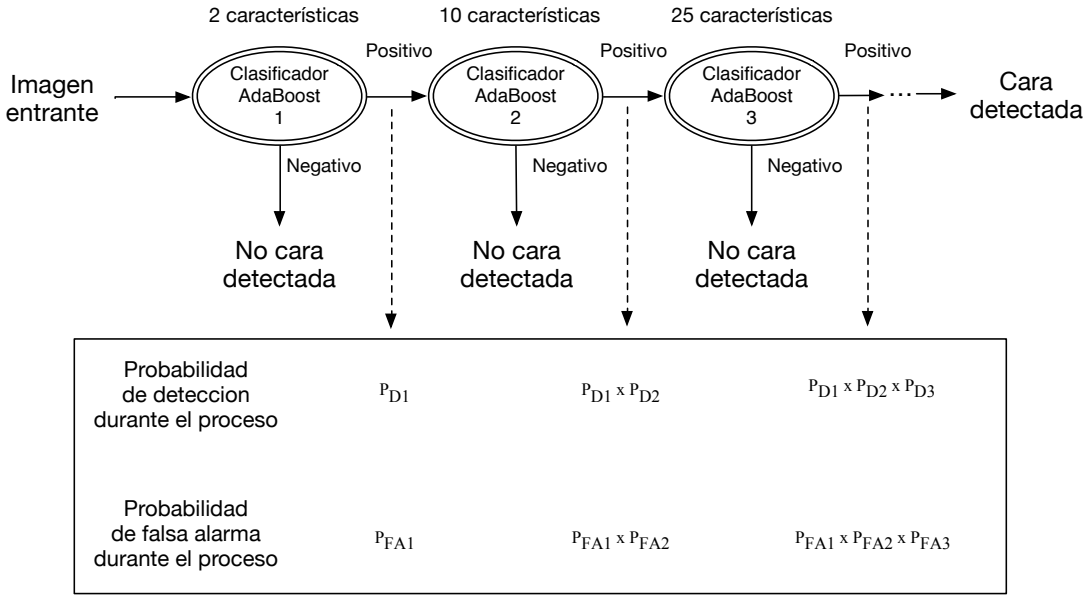


Figura 3.7 Clasificadores en cascada.

Si, por ejemplo, las probabilidades de detección de 3 clasificadores en cascada fueran 0.99 y las probabilidades de falsa alarma fueran 0.5, las probabilidades de detección y de falsa alarma después de resultar positivo el tercer clasificador serían **9.97** (0.99^3) y **0.125** (0.5^3) respectivamente, demostrando la reducción de la probabilidad de falsa alarma.

3.1.1.4 Procedimiento

En cada ventana se empieza evaluando la primera etapa del clasificador, que determina si existe la posibilidad de que haya una cara. Si el resultado de la etapa es negativo se finaliza la detección en esa ventana, lo que significa que no existe una cara en dicha zona. En caso contrario se procede de igual manera a evaluar la siguiente etapa del clasificador, y así sucesivamente hasta alcanzar la última etapa; y si el resultado de la ultima etapa es positivo se habrá detectado una cara. Este procedimiento significa un ahorro considerable de tiempo, pues no se procesan ventanas de la imagen que con certeza no contenga una cara.

La figura 3.8 muestra el diagrama de bloques general del proceso de detección:

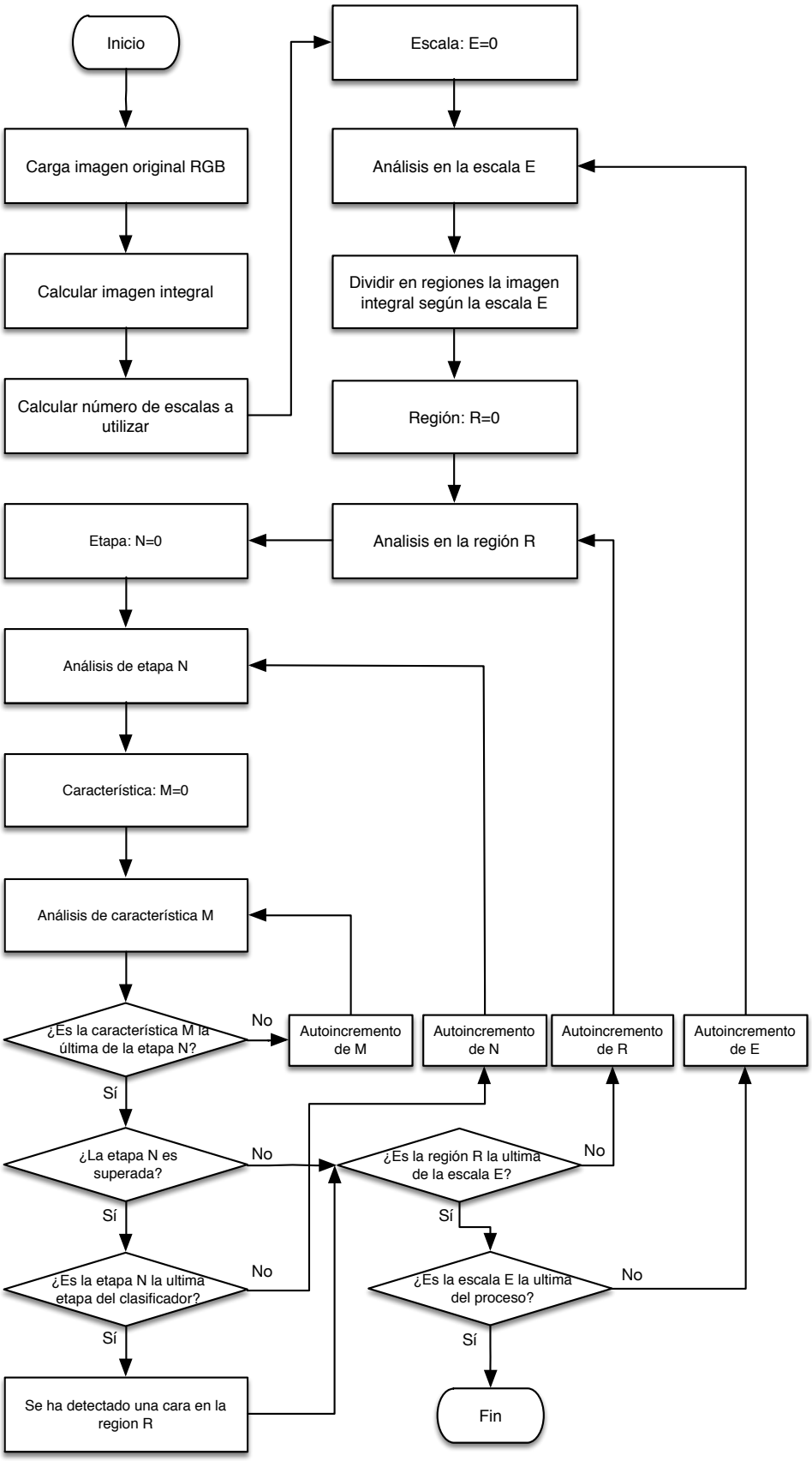


Figura 3.8 Diagrama de bloques del proceso de detección. Algoritmo de VJ.

3.2 Implementación software sobre FPGA del algoritmo de Viola-Jones

El software para la detección de caras está escrito en C/C++, mediante el entorno de desarrollo Eclipse. Altera provee una capa de software básico para el procesador Nios II, el HAL (Hardware Abstraction Layer). Esta capa permite crear una clara distinción entre la aplicación y el software del dispositivo, además ofrece servicios tales como descriptores de archivos, control de entrada-salida (E/S) y buffers.

Para llevar a cabo el objetivo del proyecto (véase 1.2), realizaremos el apartado software de acuerdo con las siguientes especificaciones:

- La imagen a mostrar mediante el puerto VGA de la placa será alojada en la memoria SRAM (véase 3.1.1.3.3). Además se alojará otra imagen en la misma memoria para intercambiar búffers entre si, consiguiendo evitar el parpadeo que produce escribir datos en la imagen. La resolución será de 640x480 píxeles con una profundidad de color de 16 bits.

- La imagen entrante de la cámara será alojada en la memoria embebida MEM_CAM (véase 3.2.1.6) con una resolución de 320x240 pixeles y una profundidad de color de 16 bits.

- La aplicación permite la detección de caras de la imagen entrante de la cámara, además de modificar algunos parámetros del proceso de detección.

- La aplicación tendrá una interfaz gráfica para facilitar su uso.

- Los datos del clasificador, así como las imágenes auxiliares para la interfaz gráfica serán alojados en la memoria FLASH (véase 3.1.1.3.2).

3.2.1 Entorno de desarrollo

El entorno de desarrollo Eclipse es un software libre multiplataforma que permite programar principalmente en C/C++ y Java. Eclipse ha sido modificado

mediante un *plugin* de Altera para ser utilizado con el procesador Nios II. Este *plugin* de eclipse nos permite incluir diversos *drivers*, aparte de los de los componentes del SoC, que incrementan la funcionalidad del sistema, como por ejemplo, un controlador para la memoria flash a nivel de software que proporciona un acceso de los ficheros de esta memoria directo desde C/C++. Otra característica de este entorno es la amplia variedad de parámetros que podemos configurar. Algunos de estos parámetros indican donde se alojarán los datos que requiera el programa (región *heap*), donde se alojará el fichero del programa, o como se manejarán las instrucciones personalizadas del procesador (en este proyecto, las instrucciones de punto flotante), entre otros.

Además de las funciones básicas del lenguaje C/C++, se incluyen librerías ya desarrolladas por los proveedores de los componentes del SoC, como Terasic (fabricante de la placa de desarrollo) o Altera (fabricante de la FPGA). La utilización de estas funciones y de las funciones desarrolladas en este proyecto serán detalladas en los siguientes apartados. El código completo está incluido en el Anexo II.

3.2.2 Configuración del sistema

El entorno Eclipse para el procesador Nios II nos permite la configuración de diversos parámetros que sirven para optimizar algunas opciones, como la habilitación de instrucciones personalizadas o la incorporación de drivers. En la tabla 3.1 muestra los parámetros configurados más relevantes del sistema, así como su descripción.

Parámetro	Configuración	Descripción
STDIN	Puerto: JTAG_UART	Establece la entrada de datos de consola por el puerto JTAG_UART
STDOUT	Puerto: JTAG_UART	Establece la salida de datos de consola por el puerto JTAG_UART (por ejemplo con el comando printf()).
Reloj del sistema	Temporizador: TIMER_MS	Indica que el reloj del sistema contará en milisegundos mediante el temporizador especificado.

Reloj de *Timestamp* (marca de tiempo).	Temporizador: TIMER_US	Indica que el reloj de *Timestamp* contará en microsegundos mediante el temporizador especificado.
Driver altera_ro_zips	Dirección de mem.: 0x8800000 Punto de montaje: "/mnt/flash/" *Offset*: 0x0	Este driver permite montar un archivo comprimido ZIP proveniente de una dirección de memoria, en este caso alojado en la memoria FLASH. Es utilizado para la carga de los recursos necesarios para la aplicación.
Región de memoria *stack*	Memoria: SDRAM	Indica la región de memoria *stack* utilizada para el apilamiento de datos relacionados con variables locales y funciones ejecutadas.
Región de memoria *heap*	Memoria: SDRAM	Indica la región de memoria *heap* utilizada para almacenar datos que requieren explícitamente memoria (función *malloc()* en lenguaje C/C++).
Región de memoria del programa	Memoria: SDRAM	Indica la región de memoria donde se almacena el texto compilado del programa a ejecutar.
Instrucciones personalizadas para variables de punto flotante.	Habilitado	Indica al compilador que el sistema contiene instrucciones personalizadas de punto flotante proporcionadas por Altera.

Tabla 3.1 Configuración del sistema software.

3.2.3 Programa

3.2.3.1 Inicialización de los recursos

El programa necesita inicializar algunos de los recursos que han sido incorporados en el SoC antes de ser utilizados, tales como el LCD, el buffer de caracteres, el puente DMA de la VGA, la cámara D5M y las interrupciones.

La clase desarrollada, "sistema", contiene las variables que hacen referencia a estos recursos para ser utilizados con las funciones de los controladores específicos, además de inicializarlos (véase Anexo II).

Las interrupciones configuradas específicamente son las del temporizador de milisegundos (TIMER_MS) que nos permite medir el tiempo transcurrido en un periodo especificado; en este caso se ha utilizado para gestionar los frames por segundo (FPS) de la interfaz VGA y de la interrupción de la entrada de datos del control remoto (REMOTE_IRQ).

Implementación:

Función	Descripción	Origen
alt_up_character_lcd_open_dev	Devuelve un puntero del tipo de variable "alt_up_character_lcd_dev" que especifica los datos generales recurso.	Altera. Controlador de LCD (SoC)
alt_up_character_lcd_init	Inicializa el recurso LCD especificado.	Altera. Controlador LCD (SoC)
alt_up_pixel_buffer_dma_open_dev	Devuelve un puntero del tipo de variable "alt_up_pixel_buffer_dma_dev" que especifica los datos generales recurso.	Altera. Controlador de VGA_DMA (SoC)
alt_up_char_buffer_open_dev	Devuelve un puntero del tipo de variable "alt_up_char_buffer_dev" que especifica los datos generales recurso.	Altera. Controlador VGA_CHAR_BUFFER (SoC)
alt_up_char_buffer_init	Inicializa el recurso VGA_CHAR_BUFFER especificado.	Altera. Controlador VGA_CHAR_BUFFER (SoC)

alt_up_char_buffer_clear	Limpia de caracteres el VGA_CHAR_BUFFER especificado.	Altera. Controlador VGA_CHAR_BUFFER (SoC)
alt_up_av_config_open_dev	Devuelve un puntero del tipo de variable “alt_up_av_config_dev” que especifica los datos generales recurso.	Altera. Controlador CAM_CONFIG (SoC)
alt_up_av_config_reset	Resetea la configuración del recurso CAM especificado.	Altera. Controlador CAM_CONFIG (SoC)
alt_ic_isr_register	Permite la activación y vinculación a una función dada, de una interrupción especificada.	Altera. Procesador Nios II

Tabla 3.2 Funciones utilizadas para la inicialización del sistema.

3.2.3.2 Gestión la interfaz VGA

Las señales de control necesarias para la interfaz VGA ya son proporcionadas por el controlador VGA_CONTROLLER automáticamente (véase 3.2.1.25), pero es necesaria una gestión entre los dos buffers proporcionados por la memoria SRAM (véase 3.2.1.5) basado en la técnica del doble buffer para evitar que se vea la línea de barrido al reescribir la imagen a emitir. La técnica del doble buffer se utiliza para eliminar los artefactos como el parpadeo o la visibilidad del barrido. El problema de la visibilidad del barrido viene dado por la dificultad de escribir los datos de una nueva imagen a mostrar antes de que la interfaz VGA vuelva a refrescar la imagen del buffer, lo que nos mostraría, por ejemplo, la mitad de la nueva imagen y la mitad de la imagen anterior. Esta técnica consiste en gestionar los dos buffer de modo que la interfaz VGA reciba la imagen de un buffer (mediante el puente DMA, véase 3.2.1.20), mientras en el otro buffer se escribe la nueva imagen. Cuando se finalice la escritura se indica al puente VGA_DMA (en este caso), que reciba los datos a mostrar del buffer que acaba de ser actualizado. Las siguientes escrituras procederán de la misma manera, alternando entre los dos buffers como muestra la figura 3.9.

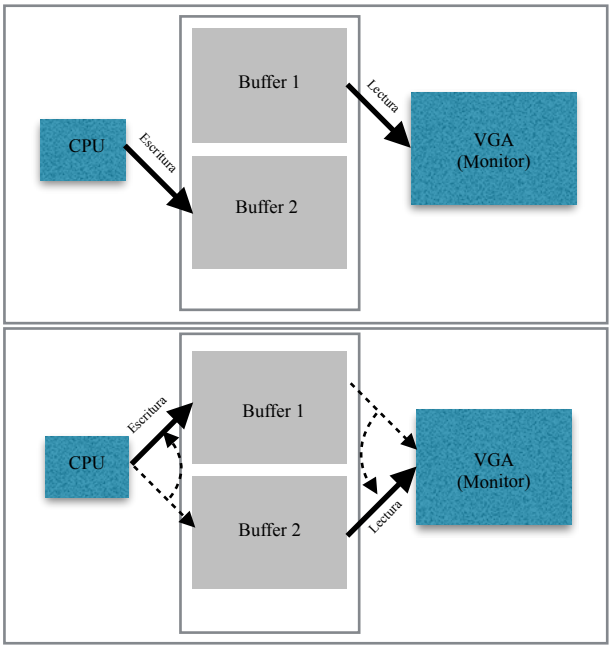


Figura 3.9 Técnica del doble buffer.

Implementación:

En la implementación se utilizan funciones básicas del lenguaje C/C++ y funciones de los propios controladores mostrados en la tabla 3.3:

Función	Descripción	Origen
memset	Establece a partir de una dirección dada, un número de bytes con un valor determinado. Es utilizado para borrar el buffer inactivo de manera más rápida que si se hiciera píxel por pixel.	Librería C/C++
dibujar	Función local de la clase "imagen" desarrollada para copiar una imagen en un buffer determinado.	Desarrollada. Clase "imagen" (véase Anexo II)
alt_up_pixel_buffer_dma_swap_buffers	Función que intercambia el puntero del buffer de lectura.	Altera. Controlador VGA_DMA
alt_up_pixel_buffer_dma_swap_buffers	Función que retorna '1' si no se ha completado el proceso de intercambio de buffers, y '0' si este ha finalizado.	Altera. Controlador VGA_DMA

Tabla 3.3 Funciones utilizadas para la gestión de los buffers VGA.

3.2.3.3 Gestión de imágenes

Para el manejo de las imágenes se ha desarrollado una clase denominada "imagen" que permite la carga de imágenes comprimidas JPEG, apoyándose en la librería de uso publico "libjpeg" desarrollado por "Independent JPEG Group". Esta clase almacena los datos de posición (en la pantalla) y tamaño de la imagen, así como su matriz de píxeles. La matriz de píxeles se compone de un array tipo "unsigned short" (16 bits, que corresponde con la profundidad de color configurada) de doble puntero para poder redimensionarse en tiempo de ejecución (ya que, de antemano, no sabemos el tamaño de la imagen a cargar). Por otro lado, se han desarrollado funciones locales que permiten la impresión de dicha imagen en un buffer dado (MEM_VGA).

Implementación:

Función	Descripción	Origen
jpeg_decode	Permite la descodificación de archivos JPEG	Librería "jpeglib" de Independent JPEG Group, modificado para la adaptación al proyecto.
fopen	Permite la apertura de un archivo cualquiera en un punto de montaje especificado.	Librería C/C++
dibujar	Función local de la clase que permite la impresión (escritura) de la imagen en un buffer de la VGA especificado.	Desarrollada. Clase "imagen" (véase Anexo II)
memcpy	Función que permite la copia de un rango de bytes de una dirección de la memoria a otro. En este caso copiamos el array que contiene los datos de imagen al buffer especificado en la función "dibujar".	Librería C/C++

Tabla 3.4 Funciones utilizadas para el manejo de imágenes.

3.2.3.4 Gestión de caracteres de la VGA

El componente VGA_CHAR_BUFFER incluye un controlador bastante simple pero completo que nos permite borrar cadenas e imprimirlas en pantalla en una posición dada. Una vez inicializados los recursos, la clase "sistema" contiene las referencia/ puntero a este recurso mediante una variable, por ello, y para no ser necesario hacer

referencia al recurso se han implementado las funciones “vga_escribir_cadena” y “vga_borrar_cadenas” en la propia clase “sistema”.

Implementación:
La implementación ha sido apoyada por las siguientes funciones descritas en la tabla 3.5.

Función	Descripción	Origen
alt_up_char_buffer_string	Permite la impresión de una cadena de texto en la pantalla, dado un recurso CHAR_BUFFER especificado.	Altera. Controlador VGA_CHAR_BUFFER
alt_up_char_buffer_clear	Permite el borrado de todas las cadenas de texto impresas en pantalla por VGA_CHAR_BUFFER.	Altera. Controlador VGA_CHAR_BUFFER

Tabla 3.5 Funciones utilizadas para el manejo de caracteres de la VGA.

3.2.3.5 Gestión de la pantalla LCD

La pantalla LCD, al igual que el buffer de caracteres de la VGA, se inicializa en la clase “sistema”, lo que conlleva que la referencia/puntero del recurso se almacene en dicha clase. Por ello, y para simplificar el código, se ha desarrollado una simple función que nos permite escribir en dicha pantalla.

Implementación:
La implementación ha sido apoyada por las funciones descritas en la tabla 3.6.

Función	Descripción	Origen
alt_up_character_lcd_set_cursor_pos	Posiciona el cursor en el LCD especificado en una coordenada (x,y) dada.	Altera. Controlador LCD
alt_up_character_lcd_string	Permite la escritura en el LCD de una cadena de texto.	Altera. Controlador LCD

Tabla 3.6 Funciones utilizadas para el manejo de la pantalla LCD.

3.2.3.6 El proceso de detección

El proceso de detección se divide en 4 partes:

- La primera etapa es la **carga de los recursos**, es decir, la carga del clasificador desde la memoria **FLASH**.
- La segunda etapa es la **configuración** de los parámetros del usuario.

- La tercera etapa es la aplicación del **algoritmo de Viola-Jones** sobre la imagen captada.
- La cuarta y última fase **proporciona las coordenadas** sobre la imagen de la cara encontrada, si la hubiera.

3.2.3.6.1 Carga del clasificador

El clasificador AdaBoost utilizado ya ha sido entrenado para la detección de caras y pertenece a la biblioteca libre **OpenCV**, de Intel.

```
<stages>
<_>
<trees>
<_>
<_>
<feature>
<rects>
<_>3 7 14 4 -1.</_>
<_>3 9 14 2 2.</_>
</rects>
</feature>
<threshold>4.0141958743333817e-003</threshold>
<left_val>0.0337941907346249</left_val>
<right_val>0.8378106951713562</right_val>
</_>
</_>
<_>
<_>
<feature>
<rects>
<_>1 2 18 4 -1.</_>
<_>7 2 6 4 3.</_>
</rects>
</feature>
<threshold>0.0151513395830989</threshold>
<left_val>0.1514132022857666</left_val>
<right_val>0.7488812208175659</right_val>
</_>
</_>
<_>
<_>
<feature>
<rects>
<_>1 7 15 9 -1.</_>
<_>1 10 15 3 3.</_>
</rects>
</feature>
<threshold>4.2109931819140911e-003</threshold>
<left_val>0.0900492817163467</left_val>
<right_val>0.6374819874763489</right_val>
</_>
</_>
</trees>
<stage_threshold>0.8226894140243530</stage_threshold>
<parent>-1</parent>
</_>

<!——SIGUIENTES ETAPAS———>
</stages>
```

Figura 3.10 Fragmento del clasificador en formato XML.

Está almacenado en un fichero con formato **XML**, por lo que resulta necesaria una librería externa que nos permita leer este tipo de archivos; en este caso se ha utilizado la librería de código abierto **RapidXML** (de Marcin Kalicinski) debido a su bajo consumo de recursos y simplicidad. La figura 3.10 muestra la primera ***etapa*** del clasificador en **morado**, en **azul** el conjunto de las características de la etapa (etiqueta ***trees***) y en **verde** los datos de cada característica.

Los valores de los elementos *rect* de las características indican las coordenadas y el peso del elemento. El peso de cada *rect* es el multiplicador del valor proporcionado por la región dada en la imagen integral. La figura 3.11 muestra los datos de las etiquetas *rect* de las características representados sobre la imagen original.

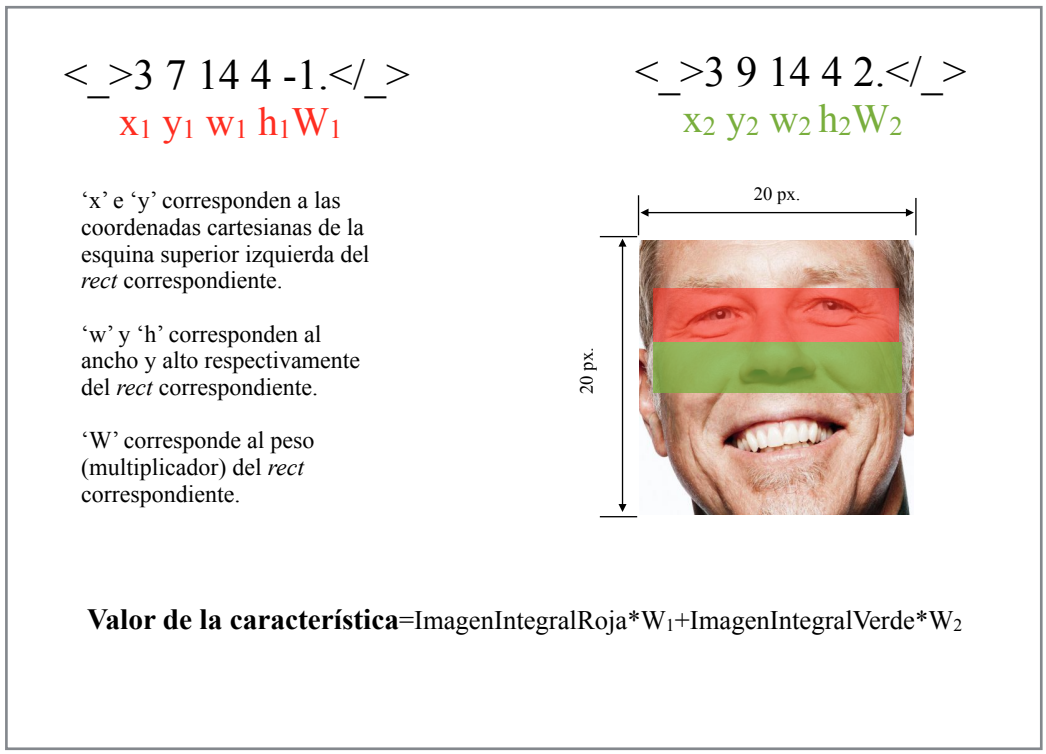


Figura 3.11 Representación de los datos de las características sobre la imagen.

Es importante tener en cuenta que todas las características de este clasificador están diseñadas para ser utilizadas en ventanas de **20x20 píxeles**, lo que significa que necesitamos escalar las coordenadas y los pesos para identificar caras en ventanas de mayor tamaño.

El resto de valores del clasificador son:

- ***Threshold***: Umbral de cada característica que es comparado con las diferencias de luminancia entre las regiones de esa característica y que determina el valor devuelto a la etapa actual.
  - ***Left_val***: Valor devuelto si la diferencia de luminancia es menor que el umbral.
  - ***Right_val***: Valor devuelto si la diferencia de luminancia es mayor que el umbral.
- ***Stage_threshold***: Umbral de cada etapa que es comparado con la acumulación de la suma de valores devueltos por las características de esa etapa y determina puede existir una cara (continua con la siguiente etapa) o definitivamente no (finalización de la detección).

Para cargar todos los datos del clasificador, se han desarrollado una serie de clases para cada elemento de la jerarquía del clasificador. Estas clases se denominan **haarcascade, stage, tree y rect**, acorde a la información que representa. La clase "haarcascade" integra un vector de variables de tipo *stage;* a su vez cada variable de tipo *stage* incorpora un vector de variables de tipo *tree*, que a su vez incorpora las variables de tipo *rect*; con lo que se consigue organizar todo el clasificador para su optima utilización en el algoritmo de Viola-Jones.

3.2.3.6.2 Parámetros configurables

Se han implementado una serie de parámetros que el usuario puede configurar para modificar la relación entre tiempo de procesamiento y probabilidad de detección. Estos parámetros **permiten manejar el número de escalas empleadas, de etapas y de ventanas utilizadas por cada escala**; estos números se expresan siempre en porcentaje con respecto al máximo utilizado en el algoritmo estándar. Por ejemplo, el número máximo de etapas empleadas por el clasificador en este caso serían 22 que corresponderían al 100% de etapas.

3.2.3.6.3 Detección

En las ecuaciones descritas de este apartado no se definen los parámetros del usuario, ya que se muestran de forma genérica similares a si el usuario opta por un análisis completo (valores de todos los parámetros del **100%**).

La fase de detección se realiza una vez configurados los parámetros, cargado el clasificador y hallada la imagen integral (véase 2.4.1). Para ello, se determina el número de escalas a utilizar y el número de ventanas a analizar en cada escala, comenzando la detección con la escala más grande, es decir, con las ventanas que más imagen abarcan, dando lugar a una primera detección de caras mas próximas a la cámara. Tras el análisis con la escala más grande, se analizan progresivamente con escalas más pequeñas hasta llegar a la escala mínima. La figura 3.12 muestra un ejemplo de este proceso.

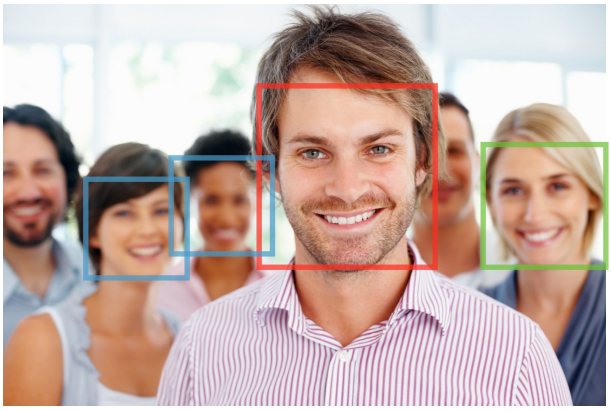


Figura 3.12 Análisis mediante diferentes escalas.

En la figura anterior se muestran diferentes caras de distinto tamaño, siendo la de mayor tamaño la marcada en rojo, y por consiguiente se detectaría antes que el resto de caras. La siguiente cara detectada sería la marcada en verde, y se finalizaría la detección con dos caras más, marcadas en azul y con el mismo tamaño (se han detectado en la misma escala).

Para determinar el número de escalas a utilizar, se hace uso de la siguiente ecuación logarítmica, cuyos parámetros vienen dados por la dimensión más pequeña de la imagen (**z**), ancho o alto, y por una constante de actualización (**x**) que puede variar entre 0 y 1 para optimizar el número de escalas. La constante de actualización determina la cantidad de escalas de forma logarítmica, es decir, cercano a 1 correspondería con infinitas ventanas (no viable) y cercano a 0 con ninguna ventana (tampoco viable).

$$\text{Número de Escalas} = -(\text{Log}(z) / \text{Log}(x))$$

En la figura 3.13 se muestra una gráfica donde **'y'** es el número de escalas a utilizar, **'x'** la constante de actualización y **'z'** la dimensión menor de la imagen a analizar.

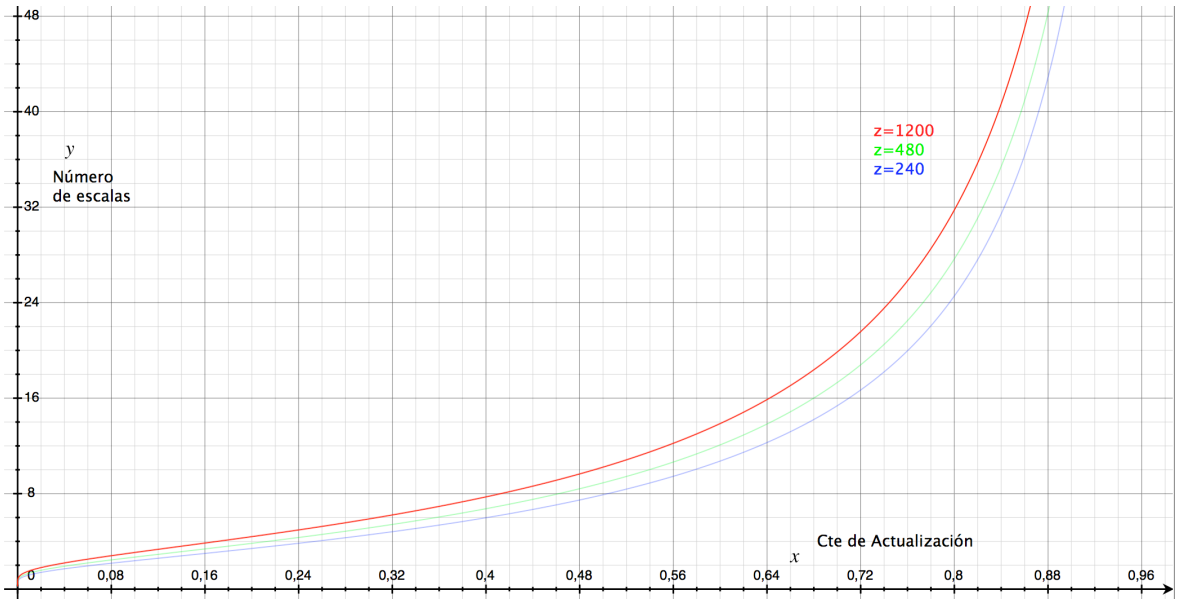


Figura 3.13 Comparativa de número de escalas con diferentes tamaños de imagen y constantes de actualización.

La imagen entrante de la cámara de este proyecto es de **320x240 pixeles**, por lo que z=240 (curva azul). La constate de actualización utilizada en este proyecto es de 0,83 ya que da lugar a aproximadamente 30 escalas, suficientes para determinar caras a distintas distancias, sin comprometer demasiado los costes de detección. El parámetro definido por el usuario se aplica al final, después de hallar el número de escalas, siendo en este caso 30 escalas si el usuario especifica el 100% de escalas, o 15 si se especifica un 50%.

Una vez hallado el número de escalas a utilizar, se halla el multiplicador que escala todos los elementos del clasificador. Para hallarlo nos basamos en la siguiente ecuación:

$$\text{Escala} = z * x^{(\text{Número de escala}-1)}$$

Donde **z** es la dimensión más pequeña de la imagen, **x** la constante de actualización y el número de escala determina el número de escala a procesar. Por ejemplo, como hemos visto, en el caso de la imagen de la cámara **z=240** y **x=0,83** por lo que si queremos hallar la escala número 20 tendríamos que:

$$\text{Escala} = 240 * 0{,}83^{(20-1)} = 6{,}96$$

Merece la pena mencionar que no todas las escalas se procesan ya que pueden dar lugar a tamaños de ventana mayores que la propia imagen, esto es debido a que al hallar las escalas no se conoce de antemano el tamaño de referencia de la ventana del clasificador. Las escalas serían válidas si el tamaño de referencia del clasificador fuera de 1x1 píxeles.

Con la escala redimensionamos la ventana de referencia del clasificador con la siguiente ecuación:

AnchoVentana = AnchoReferencia * Escala
AltoVentana = AltoReferencia * Escala

Una vez hallado el tamaño de la ventana de detección, es necesario hallar un número óptimo de ventanas/subregiones que se emplearán en la detección a la escala actual. Lo más efectivo sería que la ventana de detección se desplazara un píxel cada vez que se verifica si existe una cara o no, pero computacionalmente es muy costoso e incrementa el tiempo de detección. Por ello es necesario determinar un número óptimo de subregiones que se van a procesar, que dependa del ancho de la ventana de esa escala. Para determinar el número de subregiones que se van a analizar con un tamaño de ventana determinado utilizamos la siguiente ecuación:

SubregionesEjeX = (AnchoImagen-AnchoVentana) / Escala
SubregionesEjeY = (AltoImagen-AltoVentana) / Escala
SubregionesTotales = SubregionesEjeX * SubregionesEjeY

Con los datos hallados, el siguiente paso es escalar las características al tamaño de ventana que estamos analizando. Este proceso se realiza cada vez que se cambia de escala.

Por lo que siendo:

- $\boldsymbol{ancho_o}$: Ancho del tamaño de la característica original de referencia. En este clasificador es de 20 píxeles. En este clasificador el ancho de la ventana de referencia es de 20 píxeles.
- $\boldsymbol{alto_o}$: Alto del tamaño de la característica original dentro de la ventana de referencia. En este clasificador el alto de la ventana de referencia es de 20 píxeles.
- $\boldsymbol{x_o}$: Coordenada x de la posición de la esquina superior izquierda de la característica original.

- $\mathbf{y_o}$: Coordenada *y* de la posición de la esquina superior izquierda de la característica original.
- ***ancho***: Ancho del tamaño de la característica una vez escalado.
- ***alto***: Alto del tamaño de la característica una vez escalado.
- ***x***:Coordenada *x* de la posición de la esquina superior izquierda de la característica escalada.
- ***y***:Coordenada *y* de la posición de la esquina superior izquierda de la característica escalada.
- ***esc***: Escala actual.

Tenemos que:

$$\begin{aligned} \mathbf{ancho} &= esc*ancho_o \\ \mathbf{alto} &= esc*alto_o \\ \mathbf{x} &= esc*x_o \\ \mathbf{y} &= esc*y_o \end{aligned}$$

La detección comienza analizando la región de la primera característica de manera que siendo:

- $\boldsymbol{ValFeature_{ij}}$: Valor devuelto por la característica *i* de la etapa *j*.
- $\boldsymbol{Threshold_{ij}}$: Umbral para determinar el valor a sumar con $SumEtapa_j$ de la caracterísitica *i* de la etapa *j*.
- $\boldsymbol{SumEtapa_j}$: Valor que acumula los valores devueltos por las características de la etapa *j* tras comparar con el umbral de cada una de ellas. Sun valor inicial por etapa es 0.

El valor $Threshold_{ij}$ esta diseñado para ser utilizado con características en ventanas de 20x20 píxeles como se ha mencionado anteriormente, por lo que también es necesario escalar $ValFeature_{ij}$. Debido a que estamos escalando un área (de luminancia, dos dimensiones), el escalado de $ValFeature_{ij}$ debe ser cuadrático, es decir, dividido entre el cuadrado de la escala por el tamaño de referencia del clasificador (en este caso 20 píxeles de lado de ventana).

Por lo tanto, tenemos que:

$$\begin{aligned} ValFeature_{ij} / (esc * 20)^2 \geq Threshold_{ij} &\rightarrow SumEtapa_j' = SumEtapa_j + Right\_Val_{ij} \\ ValFeature_{ij} / (esc * 20)^2 < Threshold_{ij} &\rightarrow SumEtapa_j' = SumEtapa_i + Left\_Val_{ij} \end{aligned}$$

Una vez analizadas todas las características de la etapa se compara SumEtapa con el umbral de la etapa.

$SumEtapa_j \geq Stage\_Threshold_j \rightarrow$ Analisis de la siguiente etapa, j+1.
$SumEtapa_j < Stage\_Threshold_j \rightarrow$ Cara no encontrada. Fin de la detección.

Capítulo 4

Resultados de la evaluación

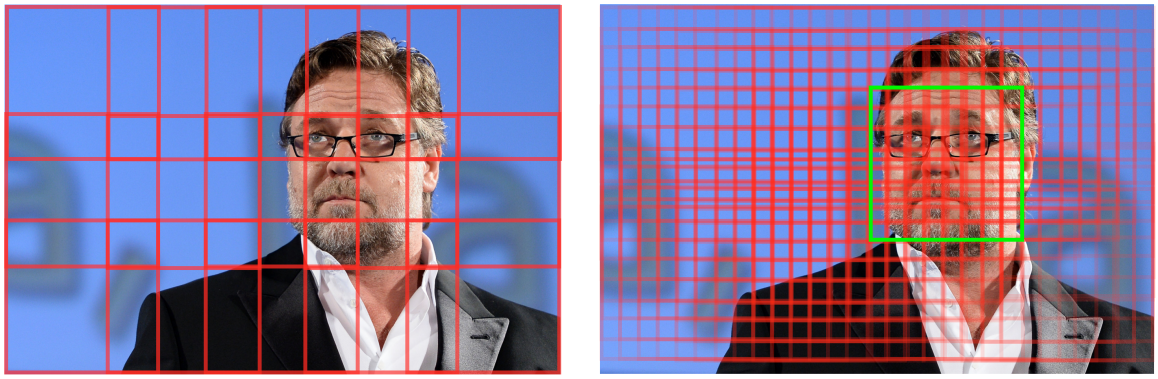
En este capítulo se realizan diversos experimentos modificando los parámetros definidos por el usuario para la evaluación del detector con un banco de imágenes que incluyen caras de diversas características, con el fin de obtener una configuración óptima. La evaluación se realiza con un conjunto de 50 fotografías donde todas contienen una cara. Los parámetros modificados para la realización del experimento son los siguientes:

- **Etapas**: Este parámetro modifica el porcentaje de etapas utilizadas por el clasificador. En este proyecto el valor 100 corresponde a 22 etapas.
- **Ventanas**: Denominamos ventanas a las sub-imagenes en las que se divide la imagen para su procesamiento. Este parámetro modifica el porcentaje de ventanas utilizadas por cada escala. El número máximo de ventanas por cada escala (valor 100) es calculado en tiempo de ejecución, y depende exclusivamente de las dimensiones de la imagen.
- **Escalas**: Este parámetro modifica el porcentaje de escalas utilizadas en el proceso de detección. El número máximo de escalas utilizadas es calculado en tiempo de ejecución, y depende al igual que el número máximo de ventanas, de las dimensiones de la imagen.

Los análisis están orientados a conseguir unos valores razonables para dichos parámetros, que mantengan una alta probabilidad de detección y baja probabilidad de falsa alarma sin que el tiempo de detección sea relativamente alto. Se modificarán los parámetros de manera independiente dejando el resto en el valor 100 (valor original). La probabilidad de falsa alarma únicamente aumentará disminuyendo el parámetro “Porcentaje de etapas”, ya que dicha probabilidad sólo depende de éste.

4.1 Modificación del número de ventanas

El detector implementado analizará cada ventana para comprobar si existe una cara o no. La modificación del número de ventanas que se analizarán se realiza mediante el parámetro "Porcentaje de ventanas", configurable por el usuario. Este parámetro modifica el número de ventanas por escala, referido al máximo de ventanas que se analizarían con el algoritmo original (valor 100 del parámetro). La figura 4.1 muestra las ventanas superpuestas para dos valores distintos de este parámetro.



a) b)

Figura 4.1 Representación orientativa de diferentes números de ventanas de detección. a) No se detecta cara. b) Se detecta cara.

Si tenemos en cuenta que el tamaño de las imágenes del conjunto de imágenes es de 320x240 píxeles (igual que la imagen entrante de la cámara), la primera escala de detección se realizará con un tamaño de ventana de 200x200 píxeles. La tabla 4.1 muestra los resultados concluyentes de los experimentos realizados con los valores de 100, 90, 80, 70, 60 y 50 para el parámetro "Porcentaje de ventanas".

Parámetro porcentaje de ventanas	Número de ventanas (1ª escala)	Probabilidad de detección	Tiempo mínimo de detección (segundos)	Tiempo máximo de detección (segundos)	Tiempo promedio de detección (segundos)
100	70	**0,84**	2,39	96,7	**34,6**
90	57	**0,84**	2,18	85,8	**31,1**
80	45	**0,84**	2,58	67,68	**26,7**
70	34	**0,72**	2,22	50,52	**20,6**
60	25	**0,64**	2,09	40,45	**15,5**
50	18	**0,64**	2,01	18,68	**9,2**

Tabla 4.1 Resultados obtenidos al variar el número de ventanas a utilizar.

La tabla anterior nos muestra una considerable disminución del tiempo de detección al disminuir el número de ventanas, pero un aumento de la probabilidad de detección al aumentar la posibilidad de que las ventanas no abarquen completamente la cara, ya que estas son equidistantes.

En la figura 4.2 se muestran las tendencias (mediante regresión exponencial) del tiempo de detección (eje Y) y el tamaño de la ventana en el que se ha detectado una cara (eje X) de cada uno de los valores del parámetro analizados.

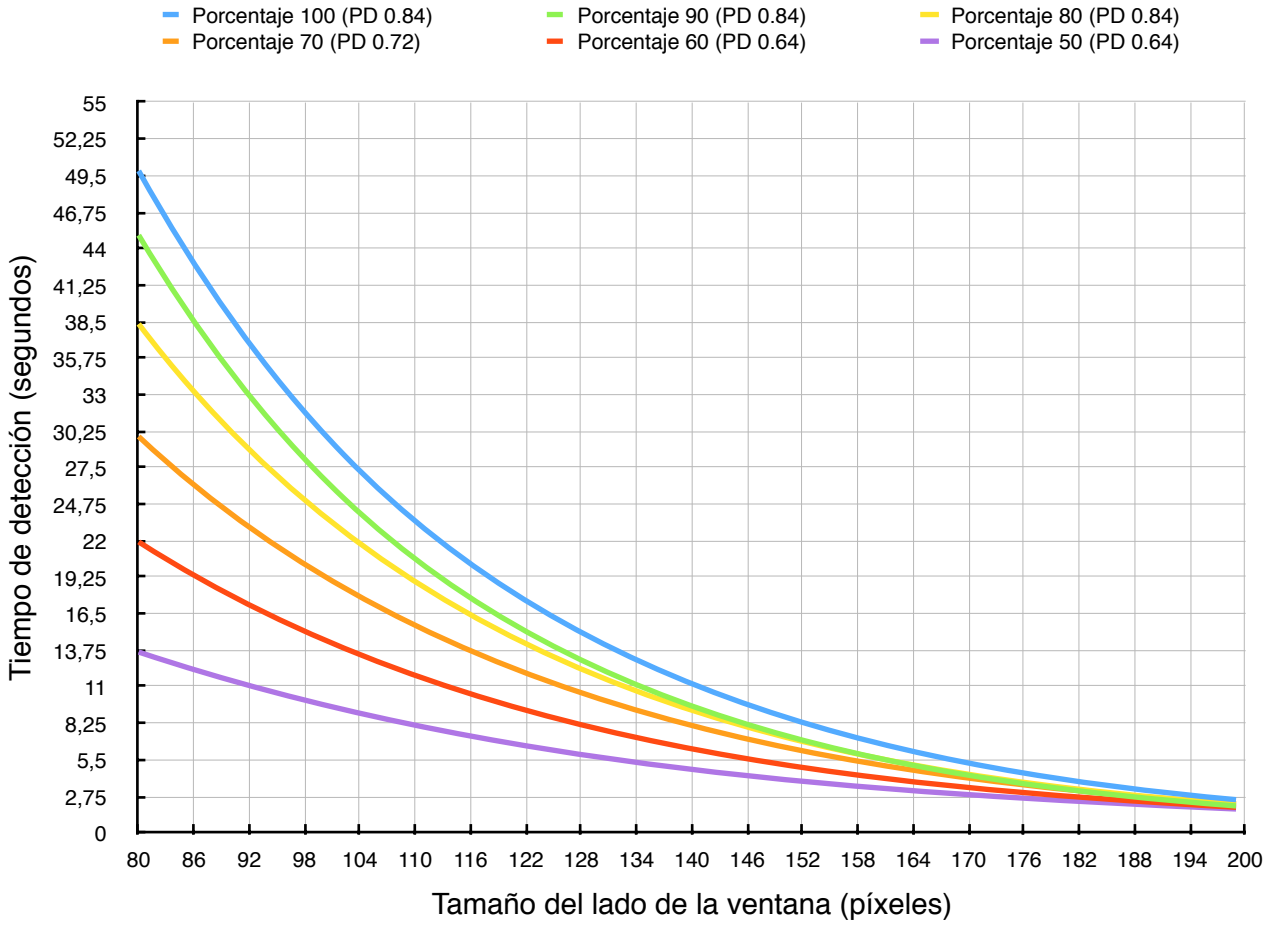


Figura 4.2 Gráfica de tendencias al variar el número de ventanas a utilizar.

Como cabría esperar, se comprueba mediante la gráfica anterior que cuanto mayor sea el número de ventanas utilizadas mayor será el tiempo de detección en ventanas de menor tamaño (caras más alejadas), pero también que este tiempo es prácticamente el mismo (aprox. 2 segundos) para caras cercanas que con el resto de valores.

4.2 Modificación del número de escalas

El número de escalas determina el número de ventanas de distinto tamaño que se van a procesar. La variación del parámetro "Porcentaje de escalas" modifica el número de ventanas en porcentaje con referencia al máximo original del algoritmo. Mediante las pruebas realizadas se ha observado que una misma cara puede ser detectada en dos o tres escalas consecutivas, por lo que en teoría se podría reducir el tiempo de detección sin influir considerablemente en la probabilidad de detección. En el experimento se ha utilizado un conjunto de 50 imágenes de 320x240 píxeles cada una (igual que la imagen entrante de la cámara). Los resultados concluyentes se muestran en la tabla 4.2.

Parámetro porcentaje de escalas	Número de escalas	Probabilidad de detección	Tiempo mínimo de detección (segundos)	Tiempo máximo de detección (segundos)	Tiempo promedio de detección (segundos)
100	14	**0,84**	2,37	95,0	**34,0**
90	12	**0,52**	2,37	81,8	**25,6**
80	11	**0,32**	2,43	80,8	**38,0**
70	10	**0,32**	2,43	80,8	**38,0**
60	7	**0,12**	2,43	15,51	**6,8**
50	6	**0,08**	2,43	15,51	**5,9**

Tabla 4.2 Resultados obtenidos al variar el número de escalas a utilizar.

Los resultados de la tabla 4.2 muestran que la probabilidad de detección disminuye considerablemente a medida que reducimos las escalas, por lo que unos valores aceptables de este parámetro serían 100 o 90 ya que a menores valores la probabilidad de detección está por debajo del 50 % de verdaderos-positivos.

En la figura 4.3 se muestran las tendencias (mediante regresión exponencial) del tiempo de detección (eje Y) y del número de escalas utilizado (eje X), que corresponde a los valores de la tabla anterior.

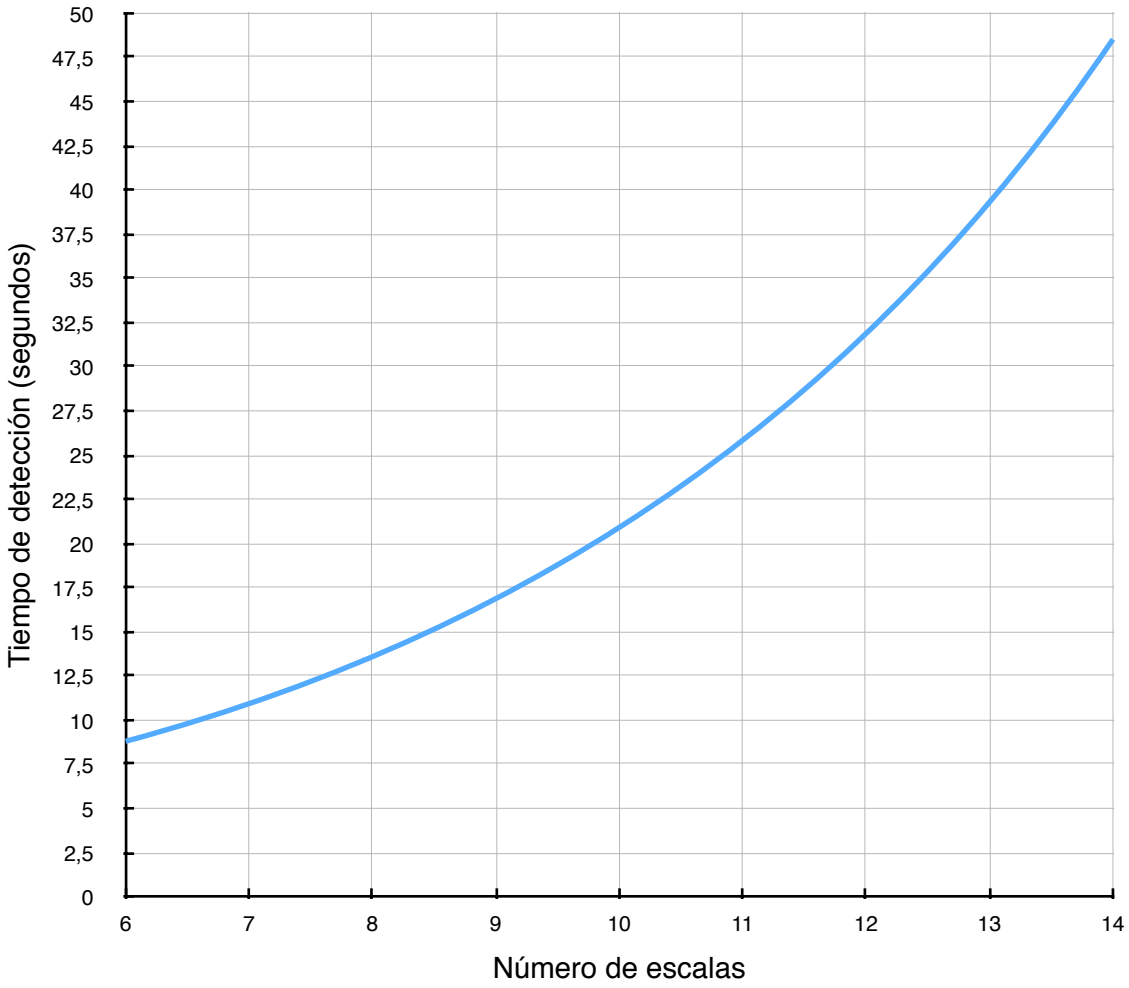


Figura 4.3 Gráfica de tendencias al variar el número de escalas a utilizar.

4.3 Modificación del número de etapas

La modificación del número de etapas que ha de superar una cara candidata permite disminuir el tiempo de detección, pero también **aumenta la probabilidad de falsa alarma**. El número máximo de etapas es 22 (100% de etapas) determinados por el clasificador utilizado. En la tabla 4.3 se muestran los resultados relevantes del experimento realizado con un conjunto de 50 imágenes de 320x240 píxeles cada una.

Parámetro porcentaje de etapas	Número de etapas	Probabilidad de detección	Probabilidad de falsa alarma	Tiempo mínimo de detección (segundos)	Tiempo máximo de detección (segundos)	Tiempo promedio de detección (segundos)
100	22	**0,84**	**0,00**	2,39	96,7	**34,4**

90	19	**0,84**	**0,00**	2,17	85,28	**30,9**
80	17	**0,8**	**0,04**	2,53	66,18	**27,2**
70	15	**0,72**	**0,04**	2,18	49,39	**20,2**
60	13	**0,6**	**0,16**	2,9	33,1	**13,5**
50	11	**0,6**	**0,2**	2,06	27,08	**10,2**

Tabla 4.3 Resultados obtenidos al variar el número de etapas a utilizar.

Se puede comprobar un aumento de la probabilidad de falsa alarma significativa, llegando al 20 % para un 50 % de las etapas. En la gráfica 4.4 se muestran las tendencias (mediante regresión exponencial) del tiempo de detección (eje Y) y del número de etapas (eje X) que corresponden con los datos de la tabla anterior.

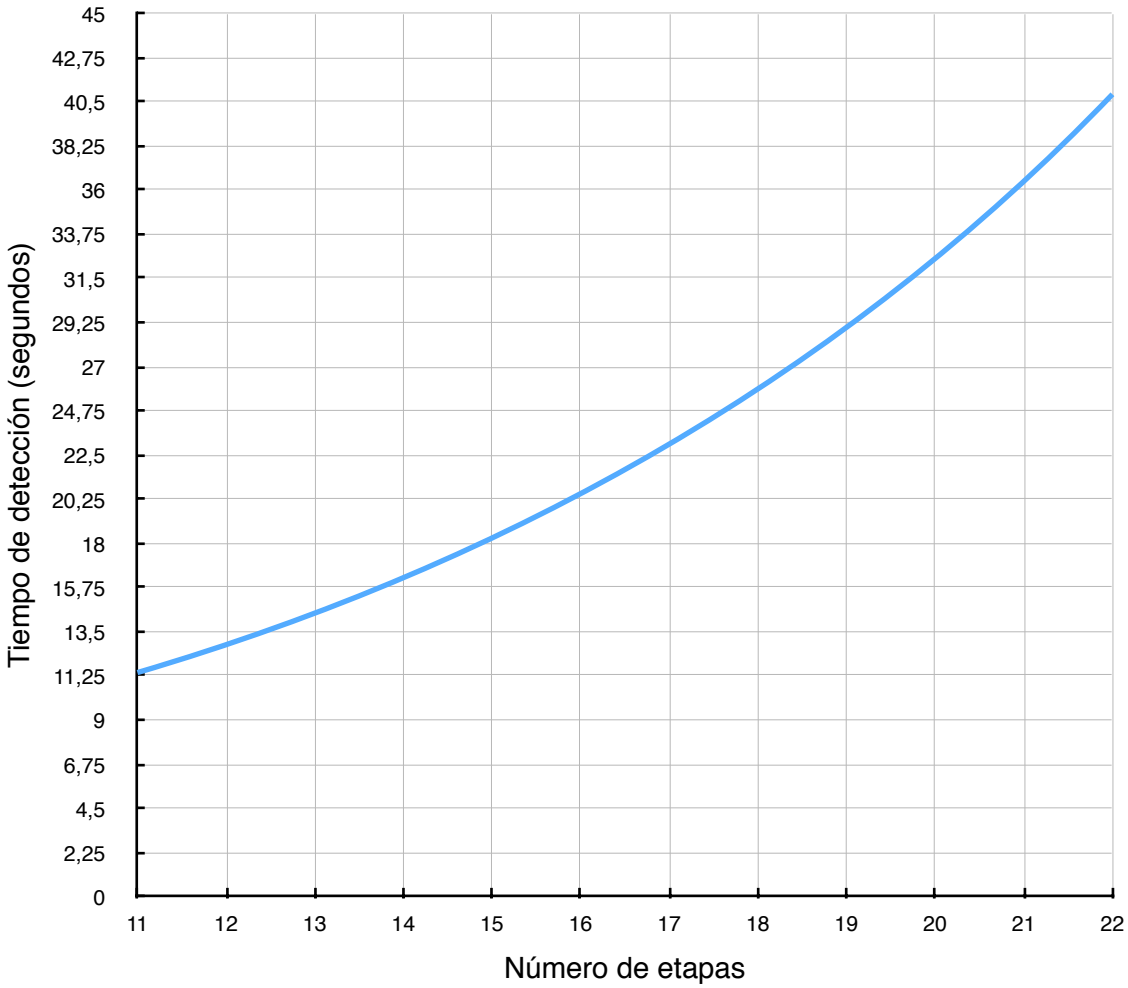


Figura 4.4 Gráfica de tendencias al variar el número de etapas a utilizar.

Los datos de la figura 4.4 junto con la tabla 4.3, muestra que a menor número de etapas (que conlleva a menor probabilidad de detección y mayor falsa alarma) menor será el tiempo de detección.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

Tras la evaluación del sistema de detección con diferentes valores de los parámetros ajustables en el capítulo 4, se ha verificado que combinando distintos valores, es posible utilizar el sistema para la detección de caras en tiempo real (aprox. 2-3 segundos) para caras cercanas, con una probabilidad de detección aceptable de aproximadamente 0.8. Para caras lejanas el sistema de detección no es útil en tiempo real por su elevado tiempo de detección, aunque si se puede utilizar para imágenes estáticas.

Los mayores problemas encontrados han sido los referentes al manejo de direcciones de memoria, a la estabilidad del procesador (debido a incompatibilidades entre las frecuencias de reloj de los componentes del SoC) y a la implementación en C++ (problemas de compatibilidad con C y C++, diversos problemas con la herramienta Eclipse no relacionados con la programación, etc.).

Por otro lado la eficacia del sistema depende mucho de la frecuencia del reloj de la CPU (relacionado directamente con las instrucciones por segundo), el cual, al estar implementado en una FPGA de bajo coste, está muy limitada en este aspecto, siendo 85 MHz la frecuencia máxima alcanzada estable. El diseño es posible exportarlo a los dispositivos ASIC (circuitos integrados de aplicación específica) que requieren un menor coste de producción. Esto es conveniente para la comercialización del producto si no se prevé ninguna modificación de hardware ya que estos dispositivos no permiten una actualización en este sentido.

En definitiva, sobre una FPGA se puede implementar cualquier tipo de aplicación que se requiera y el coste de desarrollo es relativamente bajo (dado que es reconfigurable/ borrable). En el caso de la FPGA utilizada, solo han sido utilizados un 9% de los elementos lógicos disponibles, por lo que es posible la implementación de sistemas más complejos que requieran más recursos.

5.2 Líneas futuras

Según los resultados del capítulo 4 la FPGA no es lo suficientemente potente para realizar detecciones en tiempo real. Para que fuera posible un análisis en tiempo real sería necesario un tiempo promedio de detección de 0.07 segundos mínimos para mantener la fluidez (14 fps). En la tabla 4.1 podemos observar que para una probabilidad de detección de 0.82 (aceptable) son necesarios 3 segundos para realizar el análisis en caras cercanas. Las FPGAs de tipo Cyclone IV E (la utilizada en este proyecto) tienen una tasa de MIPS (Millones de Instrucciones Por Segundo) de 181, mientras que la FPGA de gama alta Stratix V tiene una tasa de MIPS de 350 (la mayor tasa de las FPGAs de Altera) [28], lo cual llevaría a aproximadamente 1,5 segundos de tiempo de detección si se realizara en la FPGA Stratix V. Este tiempo no es suficiente para la detección en tiempo real y lo mismo ocurriría con la FPGA de Xilinx con mayor tasa de MIPS (con procesador Microblaze), la FPGA Kintex-7 con 317 MIPS [29].

- Por un lado solo un 9% de los LEs de la FPGA han sido utilizados, y por otro SOPC permite añadir más procesadores embebidos (núcleos) al sistema. Por lo tanto cabe la posibilidad de añadir más núcleos Nios II, convirtiendo la CPU en un procesador multicore capaz de realizar el proceso de detección en varias escalas paralelamente, reduciendo el tiempo de procesamiento, por ejemplo a la mitad en el caso de tener dos núcleos.

- Podría añadirse una etapa de preprocesado para normalizar el contraste y mejorar la detección, pero aumentaría el tiempo de detección.

- La piel tiene un tono de color que abarca un rango en el espacio RGB. Se podría identificar en una primera fase las posibles zonas de la piel, descartando para su análisis las zonas donde no se haya identificado piel. Este método podría mejorar sustancialmente los tiempos de detección, ya que solo una parte de la imagen incluye la cara, pudiendo reducirse en más de la mitad las ventanas a evaluar. Esto debe tomarse como una vía alternativa ya se han realizado varios experimentos que demuestran problemas con distintas razas y distintas condiciones de iluminación.

- Captación de fondo estático que permita detectar zonas en las que ha habido cambios o movimientos, para disminuir la zona de búsqueda.

- Es posible seguir una cara ya detectada, reduciendo la región de búsqueda y minimizando considerablemente el tiempo de detección e implementarlo para detección

en tiempo real. Este mecanismo resulta útil en vídeos estáticos como videoconferencias, donde la cara va a seguir en una región determinada. Sería necesario incluir un detector que determinara si es necesario analizar la imagen completa en caso de no encontrar una cara en la región donde antes si la había.

- Exportar el diseño a dispositivos ASIC (circuitos integrados de aplicación especifica), para reducir costes de producción a gran escala. El rendimiento en principio no debería de variar sustancialmente al implementarse el mismo diseño que para la FPGA.

- Exportar el diseño del software a PCs ya que estos pueden alcanzar los 9.700 MIPS (Millones de Instrucciones Por Segundo) en el caso del Intel Pentium 4 (año 2003) frente a los 181 MIPS del procesador Nios II embebido en la FPGA Cyclone IV E (la FPGA utilizada en el proyecto actual). Esto permitiría la detección en tiempo real según lo expuesto en la introducción de este apartado.

Capitulo 6

Presupuesto

Recursos materiales

Ordenador Apple Macbook Pro (vida útil 4 años)	1.600 € Duración del proyecto 1 año Amortización: 400 €
Placa de desarrollo DE2-115	420 €
Cámara D5M	79 €
Licencia iWork (Pages, Numbers, Keynote)	15 €
Monitor de pruebas	120 €
Total	**1.002 €**

Tabla 6.1 Coste de recursos materiales.

Recursos humanos

Consulta de la bibliografía	45 h
Diseño del prototipo	25 h
Programación del prototipo y pruebas	200 h
Memoria escrita	100 h
Coste ingeniero	20 €/hora
Total	**7.400 €**

Tabla 6.2 Coste de recursos humanos.

Total

Coste de recursos materiales	1.002 €
Coste de recursos humanos	7.400 €
Total	**8.402 €**

Tabla 6.3 Presupuesto total del proyecto.

Anexos

ANEXO I - Manual de usuario

Elección de parámetros

La primera parte del programa es elegir los parámetros con los que queremos realizar el análisis para detectar alguna cara. Estos parámetros permiten la elección del número de escalas, ventanas y etapas que se van a utilizar. La siguiente figura muestra una captura de pantalla de este apartado.

Figura Anexo-1.1 Aplicación. Configuración de parámetros.

A la izquierda de la figura anterior se presenta la imagen procedente de la cámara D5M que se va a analizar.

Análisis y resultados

Una vez configurado se procede al análisis. La pantalla muestra algunos de los valores del proceso, como las ventanas restantes, la escala en la cual se está analizando, etc. La siguientes figuras muestran los apartados de análisis y resultados del programa respectivamente:

Figura Anexo-1.2 Aplicación. Análisis.

Figura Anexo-1.3 Aplicación. Resultados.

Control remoto

La siguiente tabla muestra las funciones correspondientes a los botones del control remoto:

Botón	Función
1	Subir el porcentaje de etapas
2	Subir el porcentaje de ventanas
3	Subir el porcentaje de escalas
4	Bajar el porcentaje de etapas
5	Bajar el porcentaje de ventanas
6	Bajar el porcentaje de escalas
7	Bajar la ganancia de la cámara
8	Subir la ganancia de la cámara
A	Iniciar análisis
B	Volver la pantalla de configuración de parámetros y/o cancelar análisis actual.

Tabla Anexo-1.1 Controles del mando remoto.

ANEXO II - Código fuente C/C++

NOTA: En el código fuente no se incluyen las librerías RapidXML ni JPEGlib.

PROYECTO.H

```
#ifndef PROYECTO_H_
#define PROYECTO_H_
#include <vector.h>
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <sys/alt_timestamp.h>
#include <math.h>
#include <sys/alt_irq.h>
#include "lib/sistema/sistema.h"
#include "lib/deteccion/deteccion.h"
#include "lib/imagen/imagen.h"
#endif
```

PROYECTO.CPP

```
#include "proyecto.h"

int cont_tiempo_analisis;
int cont_tiempo_leds;
int  num_escalas_estado=0;
int  num_escala_actual_estado=0;
int  num_divisiones_por_escala_estado=0;
int  division_actual_estado=0;
int  tam_ventana_actual_estado=0;
unsigned int contador_vga=0;
unsigned int vga_fps=8;
unsigned int contador_estado=0;
unsigned int estado_fps=0;
unsigned short remoto;
unsigned char porcentaje_divisiones=100;
unsigned char porcentaje_escalas=100;
unsigned char porcentaje_niveles=100;
alt_u32 ganancia=40;
float  escala_actual_estado=0;
int tiempo;
bool  analizando=false;
bool  cancelar=false;
sistema * sis;
clasificador * clasif;
vector<imagen*> imagenes;
imagen * camara;
imagen * img_inicio;
imagen * img_analizando;
imagen * img_finalizado;
imagen * img_cam_analizar;
imagen * img_cam_miniatura;
volatile int remoto_cap;

/*
 * Lee el boton pulsado en el mando remoto.
 *
 * Devuelve -- El código del botón pulsado
 */
```

```
unsigned short leer_remoto()
{
	unsigned short remoto_dev = remoto;
	remoto=0;
	return remoto_dev;
}

/*
 * Imprime en pantalla los datos del clasificador
 *
 * Parámetro x -- Coordenada x en pantalla del primer caracter
 * Parámetro y -- Coordenada y en pantalla del primer caracter
 */
void detalles_clasif(int x, int y)
{
	char buffer_char [50];
	sprintf(buffer_char,"Porcentaje de etapas(1 y 4): %i%% ",
			(int)((float)clasif->etapas.size()*
				(float)porcentaje_niveles/100.f),
				porcentaje_niveles);
	sis->vga_escribir_cadena(buffer_char,x+40,y);

	sprintf(buffer_char,"Porcentaje de ventanas(2 y 5): %i%%
",
			porcentaje_divisiones);
	sis->vga_escribir_cadena(buffer_char,x+40,y+2);

	sprintf(buffer_char,"Porcentaje de escalas(3 y 6): %i%% ",
			porcentaje_escalas);
	sis->vga_escribir_cadena(buffer_char,x+40,y+4);

	sprintf(buffer_char,"Ganancia de la camara(8 y 7): %i ",
			(int)ganancia);
	sis->vga_escribir_cadena(buffer_char,x+40,y+6);

		if(analizando)
		{

			sprintf(buffer_char,"Num. escalas: %i  ",
				num_escalas_estado);
			sis->vga_escribir_cadena(buffer_char,x,y);

			sprintf(buffer_char,"Num. escala actual: %i
",
				num_escala_actual_estado);
			sis->vga_escribir_cadena(buffer_char,x,y+2);

			sprintf(buffer_char,"Escala de ventana: %.2f
",
				escala_actual_estado);
			sis->vga_escribir_cadena(buffer_char,x,y+4);

			sprintf(buffer_char,"Tam. de la ventana: %ix%i
px.",
				tam_ventana_actual_estado,
				tam_ventana_actual_estado);
			sis->vga_escribir_cadena(buffer_char,x,y+6);

			sprintf(buffer_char,"Num. subregiones: %i
",
				num_divisiones_por_escala_estado);
			sis->vga_escribir_cadena(buffer_char,x,y+9);
```

```
                  sprintf(buffer_char,"Num. subregion actual: %i
",
                              division_actual_estado);
                  sis->vga_escribir_cadena(buffer_char,x,y+11);

                  sprintf(buffer_char,"Tiempo de la deteccion:
%is        ",
                              tiempo);
                  sis->vga_escribir_cadena(buffer_char,x,y+13);


            }

}

/*
 * Interrupción que gestiona el mando remoto
 */
static void int_remoto(void* context)
{
      volatile int* remoto_cap_ptr = (volatile int*) context;
      *remoto_cap_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP
                  (REMOTE_IRQ_BASE);
      remoto=IORD(REMOTE_DATA_BASE,0);
      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(REMOTE_IRQ_BASE, 0);
      IORD_ALTERA_AVALON_PIO_EDGE_CAP(REMOTE_IRQ_BASE);

}

/*
 * Interrupción ejecutada cada 1 ms que gestiona diversos
 * componentes como el refresco de la VGA, el estado del
 * programa o distintos contadores de tiempo.
 */
static void int_temporizador_ms(void* context)
{
      contador_vga++;
      if(vga_fps>=1)
      if(contador_vga>=1000/vga_fps)
      {
            sis->vga_refresco(&imagenes);
            contador_vga=0;
      }

      contador_estado++;
      if(estado_fps>0)
      if(contador_estado>=1000/estado_fps)
      {
            detalles_clasif(5,18);
            if (leer_remoto()==REMOTE_B && analizando)
                  {
                  cancelar=true;
                  }
            contador_estado=0;



      }

      cont_tiempo_analisis++;
      if(analizando && cont_tiempo_analisis>1000)
      {
            tiempo++;
```

```
            if(tiempo>500)
            cancelar=true;
            sis->hex_escribir_segundos(tiempo);
            cont_tiempo_analisis=0;
      }

      cont_tiempo_leds++;
      if(cont_tiempo_leds>=250)
      {
            if(analizando)
            {
                  if(IORD_ALTERA_AVALON_PIO_DATA(LEDR_BASE)<1)
                  {
                        IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE,
                        262143);
                        IOWR_ALTERA_AVALON_PIO_DATA(LEDG_BASE,
                        511);
                  }
                  else
                  {
                        IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE,
                        0);
                        IOWR_ALTERA_AVALON_PIO_DATA(LEDG_BASE,
                        0);
                  }
            }
            else
            {
                  IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE,0);
                  IOWR_ALTERA_AVALON_PIO_DATA(LEDG_BASE,0);
            }
            cont_tiempo_leds=0;

      }


      IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_MS_BASE, 0);


}

/*
 * Función que genera un retardo en la ejecución
 *
 * Parámetro ms -- Milisegundos de retardo
 */
void dormir_ms(alt_u32 ms)
{
      alt_u32 t1 = alt_timestamp();
      alt_u32 t2 = alt_timestamp();
      alt_u32 max = ms*(alt_timestamp_freq()/1000);
      while(t2-t1<max)  t2 = alt_timestamp();


}

/*
 * Función que dibuja un rectángulo sobre la imagen en
 * pantalla
 *
 * Parámetro z -- Variable de tipo zona
 */
void dibujar_zona (zona * z)
```

```
{
	alt_up_pixel_buffer_dma_draw_rectangle
	(sis->vga,z->x, z->y,z->x+z->ancho,z->y+z->alto,z->color,
			0);
}

/*
 * Función que carga las imagenes necesarias para el programa
 */
void cargaImagenes()
{
	img_inicio =
			new imagen("/mnt/flash/interfaz/Inicio.jpg",
				0,0);
	img_analizando =
			new imagen("/mnt/flash/interfaz/
Analizando.jpg",
				0,0);
	img_finalizado =
			new imagen("/mnt/flash/interfaz/
Finalizado.jpg",
				0,0);
}

/*
 * Funcion que carga el clasificador en la variable global
 * "clasif"
 */
void cargarClasificador()
{
	clasif = new clasificador
		("/mnt/flash/clasificadores/
clasificador_frontal.xml");

}

/*
 * Representa en pantalla el logo de la Universidad
 * durante 2 segundos.
 */
void logo_inicial()
{
	imagen logo = imagen("/mnt/flash/interfaz/logo_uc3m.jpg",
			0,0);
	imagenes.push_back(&logo);
	dormir_ms(2000);
	imagenes.clear();
	dormir_ms(500);
}

/*
 * Función que convierte la imagen entrante de la cámara
 * en una variable de tipo "imagen"
 *
 * Devuelve una variable tipo imagen.
 */
imagen * camara_img ()
{
	int i;
	imagen * cam = new  imagen();
	cam->ancho=320;
	cam->alto=240;
	cam->x=0;
```

```
	cam->y=0;
	cam->visible=true;
	cam->img = new unsigned short  *[240];

	for(i=0;i<240;i++){
		cam->img[i]=(unsigned short*)(0x08080000+i*320*2);
	}
	return cam;
}

/*
 * Función que inicializa las interrupciones, clasificador,
 * imágenes y los distintos componentes del sistema
 */
void inicio_sistema()
{
	sistema * sis_ptr = new sistema();
	sis=sis_ptr;

	alt_ic_isr_register(TIMER_MS_IRQ_INTERRUPT_CONTROLLER_ID,
			TIMER_MS_IRQ,
			int_temporizador_ms,
		NULL,0x0);

	void* remoto_cap_ptr = (void*) &remoto_cap;
	IOWR_ALTERA_AVALON_PIO_IRQ_MASK(REMOTE_IRQ_BASE, 0xf);
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(REMOTE_IRQ_BASE, 0x0);

alt_ic_isr_register(REMOTE_IRQ_IRQ_INTERRUPT_CONTROLLER_ID,
			REMOTE_IRQ_IRQ,int_remoto,remoto_cap_ptr,0x0);

	logo_inicial();

	sis->vga_escribir_cadena("Cargando imagenes de la
interfaz..."
			,22,28);
	cargaImagenes();

	sis->vga_escribir_cadena("Cargando clasificador...",
28,30);
	cargarClasificador();
	sis->vga_borrar_cadenas();

	camara=camara_img();
}

/*
 * Función que engloba todo el apartado de detección para su
 * presentación y control
 */
void deteccion_camara()
{
	int i,j;
	zona cara;
	camara->x=31;
	camara->y=131;
	img_cam_analizar = new imagen(320,240,0,0);
	img_cam_miniatura = new imagen(160,120,50,330);
	imagenes.push_back(img_inicio);
	imagenes.push_back(camara);
	alt_up_av_config_write_D5M_cfg_register
	(sis->cam,0x35,ganancia);
	estado_fps=1;
```

```
char buffer_char [50];
sis->lcd_borrar();
sis->lcd_escribir("En espera",0,1);
while(1)
{
                  switch (leer_remoto())
                  {

                  case REMOTE_A:

                        for(j=0;j<240;j++)

                        for(i=0;i<320;i++)

                        img_cam_analizar->
                                                img[j]
                        [i]=camara->img[j][i];
                        for(j=0;j<120;j++)
                              for(i=0;i<160;i++)
                              img_cam_miniatura->img[j]
                        [i]=

                        img_cam_analizar->img[j*2][i*2];

                        imagenes.at(0)=img_analizando;
                        imagenes.at(1)=img_cam_miniatura;

                        vga_fps=1;
                        analizando=true;
                        sis->lcd_borrar();
                        sis->lcd_escribir("Analizando...",
                        0,1);
                        tiempo=0;
                        cara = vj(clasif,img_cam_analizar,
                                    porcentaje_niveles,
                                    porcentaje_divisiones,
                                    porcentaje_escalas,
                                    &num_escalas_estado,

                              &num_escala_actual_estado,
                                    &escala_actual_estado,

                        &num_divisiones_por_escala_estado,

                        &division_actual_estado,

                  &tam_ventana_actual_estado,&cancelar);


                        if(cara.alto>0 && cara.ancho>0);
                        {
                              cara.x=cara.x/
                              2+img_cam_miniatura->x;
                              cara.ancho=cara.ancho/2;
                              cara.y=cara.y/
                              2+img_cam_miniatura->y;
                              cara.alto=cara.alto/2;
                              vga_fps=14;

                        imagenes.at(0)=img_finalizado;
                              dormir_ms(500);
                              vga_fps=0;
                              dibujar_zona(&cara);
```

```
                sis->lcd_borrar();
                sis->lcd_escribir("Cara
                encontrada",
                            0,0);
                sprintf(buffer_char,"X=%i ;
                Y=%i",
                cara.x+cara.ancho/2,cara.y
                +cara.alto/2);
                sis-
                >lcd_escribir(buffer_char,
                0,1);
                sprintf(buffer_char,
                            "Cara encontrada
                            en: X=%i Y=%i",
                            (int)(cara.x
                            +cara.ancho/2),
                            (int)(cara.y
                            +cara.alto/2));
                sis->vga_escribir_cadena
                (buffer_char,5,33);
                analizando=false;
                if(cara.alto>0 &&
                cara.ancho>0)
                      while(leer_remoto()!
                      =REMOTE_B);
                }
          sis->hex_reiniciar();
          analizando=false;
          cancelar=false;
          sis->vga_borrar_cadenas();
          imagenes.at(0)=img_inicio;
          imagenes.at(1)=camara;
          sis->lcd_borrar();
          sis->lcd_escribir("En espera",
          0,1);
          vga_fps=8;
          break;

    case REMOTE_1:
          porcentaje_niveles=
                      porcentaje_niveles+5;
          if (porcentaje_niveles>100)
                porcentaje_niveles=100;
          detalles_clasif(5,18);
          break;
    case REMOTE_4:
          porcentaje_niveles=
                      porcentaje_niveles-5;
          if (porcentaje_niveles<5)
                porcentaje_niveles=5;
          detalles_clasif(5,18);
          break;
    case REMOTE_2:
          porcentaje_divisiones=
                      porcentaje_divisiones
                      +5;
          if (porcentaje_divisiones>100)
                porcentaje_divisiones=100;
          detalles_clasif(5,18);
          break;
    case REMOTE_5:
          porcentaje_divisiones=
```

```
                        porcentaje_divisiones
                        -5;
                        if (porcentaje_divisiones<5)
                              porcentaje_divisiones=5;
                        detalles_clasif(5,18);
                        break;
                  case REMOTE_3:
                        porcentaje_escalas=
                                    porcentaje_escalas+5;
                        if (porcentaje_escalas>100)
                              porcentaje_escalas=100;
                        detalles_clasif(5,18);
                        break;
                  case REMOTE_6:
                        porcentaje_escalas=
                                    porcentaje_escalas-5;
                        if (porcentaje_escalas<5)
                              porcentaje_escalas=5;
                        detalles_clasif(5,18);
                        break;
                  case REMOTE_7:
                        if (ganancia>1)
                        ganancia--;

                  alt_up_av_config_write_D5M_cfg_register
                        (sis->cam,0x35,ganancia);
                        detalles_clasif(5,18);
                        break;
                  case REMOTE_8:
                        if(ganancia<63)
                        ganancia++;

                  alt_up_av_config_write_D5M_cfg_register
                        (sis->cam,0x35,ganancia);
                        detalles_clasif(5,18);
                        break;
                  case REMOTE_POWER:
                        vga_fps=0;
                        sis->vga_borrar();
                        sis->vga_borrar_cadenas();
                        return ;
                        break;
                  }
            }
}

/*
 * Bucle principal
 *
 * Devuelve el código de salida del programa
 */
int main()
{

      inicio_sistema();
      deteccion_camara();
      delete sis;
      return 0;
}
```

SISTEMA.H

```
using namespace std;

#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include <vector.h>
#include <sys/alt_timestamp.h>
#include <sys/alt_alarm.h>
#include <altera_avalon_timer_regs.h>
#include <altera_avalon_pio_regs.h>
#include <altera_up_avalon_character_lcd.h>
#include <altera_up_avalon_video_pixel_buffer_dma.h>
#include <altera_up_avalon_audio_and_video_config.h>
#include <altera_up_avalon_audio_and_video_config_regs.h>
#include <altera_up_avalon_video_character_buffer_with_dma.h>

#include "../imagen/imagen.h"

#define CAM_M 0x08080000

#define REMOTE_A 61455
#define REMOTE_B 60435
#define REMOTE_C 61200
#define REMOTE_1 65025
#define REMOTE_2 64770
#define REMOTE_3 64515
#define REMOTE_4 64260
#define REMOTE_5 64005
#define REMOTE_6 63750
#define REMOTE_7 63495
#define REMOTE_8 63240
#define REMOTE_9 62985
#define REMOTE_0 65280
#define REMOTE_POWER 60690
#define REMOTE_VOL_UP 58395
#define REMOTE_VOL_DOWN 57375
#define REMOTE_CH_UP 58650
#define REMOTE_CH_DOWN 57630
#define REMOTE_MENU 60945
#define REMOTE_PLAY 59670
#define REMOTE_ADJ_LEFT 60180
#define REMOTE_ADJ_RIGHT 59160
#define REMOTE_MUTE 62220
#define REMOTE_RETURN 59415


/*
 * Clase que inicializa y gestiona los componentes del
 * sistema
 */

class sistema {

public:

	alt_up_character_lcd_dev * lcd;
	alt_up_pixel_buffer_dma_dev * vga;
	alt_up_av_config_dev* cam;
	alt_up_char_buffer_dev * vga_char;
```

```
  /*
   * Inicializa el sistema
   */
sistema();

/*
 * Destruye el objeto sistema
 */
  ~sistema();

  /*
   * Refresca la imagen saliente de la VGA
   *
   * Parámetro imagenes -- Vector que contiene las imágenes
   * nuevas a mostrar por orden de profundidad
   */
  void vga_refresco(vector<imagen*> *imagenes);
  /*
   * Borra la imagen saliente de la VGA
   */
  void vga_borrar();
  /*
   * Escribe una cadena en la imagen saliente de la VGA
   *
   * Parámetro ptr -- Puntero a la cadena de caracteres a
   * imprimir en pantalla
   *
   * Parámetro x -- Coordenada X de la pantalla donde se
   * inicia la cadena
   *
   * Parámetro y -- Coordenada Y de la pantalla donde se
   * inicia la cadena
   */
  void vga_escribir_cadena(const char *ptr,int x, int y);
  /*
   * Borra las cadenas de texto de la pantalla
   */
  void vga_borrar_cadenas();
  /*
   * Escribe una cadena de caracteres en la pantalla LCD
   *
   * Parámetro str -- Puntero a la cadena de caracteres a
   * escribir en el LCD
   *
   * Parámetro x -- Coordenada X del LCD donde se inicia
   * la cadena
   *
   * Parámetro y -- Coordenada Y del LCD donde se inicia
   * la cadena
   */
  void lcd_escribir(const char * str,int x, int y);
  /*
   * Borra todos los caracteres en la pantalla lcd
   *
   */
  void lcd_borrar();
  /*
   * Representa los segundos en los displays hexadecimales
   *
   * Parámetro segundos -- Segundos a mostrar
   *
   */
```

```
        void hex_escribir_segundos(int segundos);

        /*
         * Configura el valor 0 en todos los displays
         * hexadecimales
         *
         */
        void hex_reiniciar();
};
```

SISTEMA.CPP

```
#include "sistema.h"

sistema::sistema()
{
                        /*
                         * Dispositivo LCD
                         */
                        lcd = alt_up_character_lcd_open_dev
                                    (LCD_NAME);
                        alt_up_character_lcd_init(lcd);
                        alt_up_character_lcd_cursor_off(lcd);
                        lcd_escribir("Sistema",0,0);
                        lcd_escribir("creado",0,1);

                        /*
                         * Dispositivo VGA
                         */
                        vga = alt_up_pixel_buffer_dma_open_dev
                        (VGA_DMA_NAME);

                alt_up_pixel_buffer_dma_clear_screen(vga,0);

                alt_up_pixel_buffer_dma_clear_screen(vga,1);


                        /*
                         *  VGA CHAR
                         */

                        vga_char = alt_up_char_buffer_open_dev
                                    ("/dev/vga_char_buffer");
                        alt_up_char_buffer_init(vga_char);
                        alt_up_char_buffer_clear(vga_char);

                        /*
                         * CAM
                         */

                        cam = alt_up_av_config_open_dev
                                    (CAM_CONFIG_NAME);
                        cam->type=TRDB_D5M_CONFIG;
                        alt_up_av_config_reset(cam);

                        /*
                         * Iniciamos Timestamp
                         */
                        alt_timestamp_start();
}

sistema::~sistema()
```

```
{
	free(lcd);
	free(vga);
}

void sistema::vga_escribir_cadena(const char *ptr,int x,
		int y)
{
	alt_up_char_buffer_string(vga_char,ptr,x,y);
}

void sistema::vga_borrar_cadenas()
{
	alt_up_char_buffer_clear(vga_char);
}

void sistema::vga_refresco(vector<imagen*>  * imagenes)
{
	int i;

	  memset((void*)vga->back_buffer_start_address,
			0,640*480*2);

	if(imagenes->size()>0)
	{
		for(i=0;i<(int)imagenes->size();i++)
		{
			if (imagenes->at(i)!=NULL &&
					imagenes->at(i)->visible)
				{
					imagenes->at(i)->dibujar
						(vga-
					>back_buffer_start_address);
					memset((void*)
						(vga-
					>back_buffer_start_address
						+640*480*2),
						0,640*2*20);
				}
		}
	}


	  alt_up_pixel_buffer_dma_swap_buffers(vga);

while(alt_up_pixel_buffer_dma_check_swap_buffers_status(vga));

}

void sistema::vga_borrar()
{
	memset((void*)vga->back_buffer_start_address,0,640*480*2);
    alt_up_pixel_buffer_dma_swap_buffers(vga);

while(alt_up_pixel_buffer_dma_check_swap_buffers_status(vga));

}

void sistema::lcd_escribir(const char * str,int x, int y)
{

	if(x>=0 && x<16 && y>=0 && y<2)
	{
```

```
            alt_up_character_lcd_set_cursor_pos(lcd,x,y);
            alt_up_character_lcd_string(lcd,str);
      }
}
void sistema::lcd_borrar()
{
      alt_up_character_lcd_set_cursor_pos(lcd,0,0);
      alt_up_character_lcd_string(lcd,"                ");
      alt_up_character_lcd_set_cursor_pos(lcd,0,1);
      alt_up_character_lcd_string(lcd,"                ");
}
void sistema::hex_escribir_segundos(int segundos)
{
      IOWR_ALTERA_AVALON_PIO_DATA(HEX0_BASE,segundos % 10);
      IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE,segundos /10 % 10);
      IOWR_ALTERA_AVALON_PIO_DATA(HEX2_BASE,segundos /100% 10);
      IOWR_ALTERA_AVALON_PIO_DATA(HEX3_BASE,segundos /1000 %
10);

}

void sistema::hex_reiniciar()
{
      IOWR_ALTERA_AVALON_PIO_DATA(HEX0_BASE,0);
      IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE,0);
      IOWR_ALTERA_AVALON_PIO_DATA(HEX2_BASE,0);
      IOWR_ALTERA_AVALON_PIO_DATA(HEX3_BASE,0);
}
```

IMAGEN.H

```
#ifndef IMAGEN_H_
#define IMAGEN_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cstdio>
#include <io.h>
#include "jpeg/jpeglib.h"
#include "jpeg/jerror.h"
#include <altera_up_avalon_video_pixel_buffer_dma.h>

#define  ANCHO_PANTALLA 640
#define  ALTO_PANTALLA 480



class imagen
{
private:
      void jpeg_decode(FILE * infile);
public:

      unsigned short  **img;
      short ancho,alto,x,y;
      bool visible;
      imagen();
      imagen(char * dirImagen,int x_,int y_);
      imagen(char * dirImagen);
      imagen(short ancho_,short alto_,short x_,short y_);
      ~imagen();
```

```
        void dibujar(unsigned int  buffer);
};

#endif
```

IMAGEN.CPP

```
#include "imagen.h"

        imagen::imagen()
        {
                visible=false;
        };
        imagen::imagen(char * dirImagen,int x_,int y_){
                visible=true;
                x=x_;
                y=y_;
                FILE * f = fopen(dirImagen,"r");
                jpeg_decode(f);
                fclose(f);
        }
        imagen::imagen(char * dirImagen){
                visible=true;
                x=0;
                y=0;
                FILE * f = fopen(dirImagen,"r");
                jpeg_decode(f);
                fclose(f);
                }
        imagen::imagen(short ancho_,short alto_,short x_,short y_)
{
                int i;
                visible=true;
                ancho=ancho_;
                alto=alto_;
                x=x_;
                y=y_;
                img = new unsigned short  *[alto];
                        for (i=0;i<alto;i++) {
                                img[i]=new unsigned short  [ancho] ;

                        }



        }
        imagen::~imagen(){
                int i;
                for(i = 0; i < alto; i++) { delete[] img[i];}
            delete[] img;


        }


        void imagen::dibujar(unsigned int buffer){
                int i,resXder=0,resYabaj=0,resXizq=0,resYarrib=0;
                if((x+ancho)>ANCHO_PANTALLA)  resXder=x+ancho-
                ANCHO_PANTALLA;
                if ((y+alto)>ALTO_PANTALLA) resYabaj=y+alto-
ALTO_PANTALLA;
```

```
            if (x<0) resXizq=abs(x);
            if (y<0) resYarrib=abs(y);
            if((x+ancho)>0 && (y<ALTO_PANTALLA) && (y+alto)>0 &&
            (x<ANCHO_PANTALLA)) {

                  for (i=resYarrib;i<(alto-resYabaj);i++)
                        memcpy((void*)(buffer+(i
                        +y)*ANCHO_PANTALLA*2 +(x
                        +resXizq)*2),&img[i][resXizq],(ancho-
                        resXder-resXizq)*2);

            }
      }
```

DETECCION.H

```
#ifndef DETECCION_H_
#define DETECCION_H_
#define EscalaActualizacion 0.8333f
#define RLum 0.299f
#define GLum 0.587f
#define BLum 0.114f
#include "clasificador.h"
#include "zona.h"
#include "funciones_vj.h"
#include "../imagen/imagen.h"

#endif
```

ZONA.H

```
#ifndef ZONA_H_
#define ZONA_H_

class zona
{
public:
      int x,y,ancho,alto;
      unsigned short color;

};

#endif
```

CLASIFICADOR.H

```
#ifndef CLASIFICADOR_H_
#define CLASIFICADOR_H_

#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string.h>
#include "rapidxml/rapidxml.hpp"
#include "rapidxml/rapidxml_utils.hpp"
#include "rapidxml/rapidxml_print.hpp"
using std::string;
using std::cout;
```

```
using std::endl;
using std::ifstream;
using std::vector;
using std::stringstream;
using namespace rapidxml;

class caracteristica
{
 public:
 int x,y;
 int ancho, alto;
 double peso;
};
class arbol
{
 public:
 vector<caracteristica> caracteristicas;
 double umbral,valor_izq,valor_der;
};
class etapa
{
 public:

 vector<arbol> arboles;
 double umbral;
 int padre;
};

class clasificador
{
public:
      vector<etapa> etapas;
      char tamx,tamy;
      clasificador(char * dir_xml);

};
#endif
```

CLASIFICADOR.CPP

```
#include "clasificador.h"

clasificador::clasificador(char * dir_xml)
{

      xml_document<> doc;
      file<> xmlArchivo(dir_xml);
      doc.parse<0>(xmlArchivo.data());

      char * tamanio = doc.first_node()->first_node()->
                  first_node()->value();
      tamx=strtol(strtok(tamanio, " "),NULL,10);
      tamy=strtol(strtok(NULL, " "),NULL,10);

      xml_node<> * nodo_etapa = doc.first_node()->first_node()->
                  first_node()->next_sibling()->first_node();
      while(1)
      {
            etapa et ;
            xml_node<> * nodo_arbol=nodo_etapa->first_node()->
                        first_node(); //Primer arbol
            while(1) //Recorremos los arboles
```

```
		{
			arbol  ar;
			xml_node<> * nodo_caracteristica=nodo_arbol->
				first_node()->first_node()->
>first_node()->
				first_node(); //Primer Rect
			while(1) //Recorremos las caracteristicas
			{
				caracteristica  ca ;
				char * char_ca = nodo_caracteristica->
>value();

				ca.x = strtol(strtok(char_ca, " "),NULL,
10);
				ca.y = strtol(strtok(NULL, " "),NULL,
10);
				ca.ancho = strtol(strtok(NULL, "
"),NULL,10);
				ca.alto = strtol(strtok(NULL, " "),NULL,
10);
				ca.peso = strtod(strtok(NULL, "
"),NULL);

				ar.caracteristicas.push_back(ca);
				if(nodo_caracteristica->next_sibling()!
=0x0)

nodo_caracteristica=nodo_caracteristica->
					next_sibling();
				else break;

			}

			ar.umbral = strtod(nodo_arbol->first_node()->
				first_node()->next_sibling()->
>value(),
				NULL); //Añadimos Threshold
			ar.valor_izq = strtod(nodo_arbol->
>first_node()->
				first_node()->next_sibling()->
				next_sibling()->value(),
				NULL); //Añadimos LeftVal
			ar.valor_der = strtod(nodo_arbol->
>first_node()->
				first_node()->next_sibling()->
				next_sibling()->next_sibling()->
>value(),
				NULL); //Añadimos RightVal
			et.arboles.push_back(ar);

			if(nodo_arbol->next_sibling()!=0x0)
				nodo_arbol=nodo_arbol->next_sibling();
			else break;

		}
		et.umbral=strtod(nodo_etapa->first_node()->
				next_sibling()->value(),NULL);
		et.padre=strtol(nodo_etapa->first_node()->
				next_sibling()->next_sibling()->value(),
				NULL,10);
		etapas.push_back(et);
		if(nodo_etapa->next_sibling()!=0x0)
			nodo_etapa=nodo_etapa->next_sibling();
```

```
            else break;
        }
}
```

FUNCIONES_VJ.H

```
#ifndef FUNCIONES_VJ_H_
#define FUNCIONES_VJ_H_


#include <vector>
#include <math.h>
#include <sys/alt_timestamp.h>
#include "zona.h"
#include "clasificador.h"
#include "../imagen/imagen.h"

/*
 * Ejecuta la detección de caras mediante el algoritmo de
 * Viola-Jones
 *
 * Parámetro haarc -- Puntero al clasificador
 *
 * Parámetro img -- Puntero a la imagen a analizar
 *
 * Parámetro porcentaje_niveles -- Valor del porcentaje de
 * niveles a utilizar
 *
 * Parámetro porcentaje_divisiones -- Valor del porcentaje de
 * divisiones/ventanas a utilizar
 *
 * Parámetro porcentaje_escalas -- Valor del porcentaje de
 * escalas a utilizar
 *
 * Parámetro num_escalas_o -- Puntero del número de escalas
 * que se van a analizar
 *
 * Parámetro num_divisones_por_escala_o -- Puntero del número
 * de divisiones en la escala actual que se van a analizar
 *
 * Parámetro num_division_actual_o -- Puntero del número de
 * division que se está analizando
 *
 * Parámetro tam_ventana_actual_o -- Puntero del tamaño de la
 * ventana que se está analizando
 *
 * Parámetro cancelar -- Puntero que permite cancelar el
 * proceso
 */
zona vj(clasificador * clasif,imagen * img,unsigned char
      porcentaje_niveles,unsigned char porcentaje_divisiones,
      unsigned char porcentaje_escalas,  int * num_escalas_o,
      int * num_escala_actual_o,float * escala_actual_o,
      int * num_divisiones_por_escala_o, int *
division_actual_o,
      int * tam_ventana_actual_o, bool * cancelar );

#endif
```

FUNCIONES_VJ.CPP

```
#include "funciones_vj.h"
#include "deteccion.h"

bool escala_admitida(int escalas, int e)
{
	switch (escalas)
	{
	case 1:
		if (e==1)
			return true;
		break;
	case 2:
		if (e==1 || e==2)
			return true;
		break;
	case 3:
		if (e==1 || e==2 || e==3)
			return true;
		break;
	case 4:
		if (e==1 || e==2 || e==3 || e==4)
			return true;
		break;
	case 5:
		if (e==1 || e==2 || e==3 || e==4 || e==5)
			return true;
		break;
	case 6:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6)
			return true;
		break;
	case 7:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
e==7)
			return true;
		break;
	case 8:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8)
			return true;
		break;
	case 9:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8 || e==9)
			return true;
		break;
	case 10:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8 || e==9 || e==10)
			return true;
		break;
	case 11:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8 || e==9 || e==10 || e==11)
			return true;
		break;
	case 12:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8 || e==9 || e==10 || e==11 || e==12)
		return true;
		break;
```

```
	case 13:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8 || e==9 || e==10 || e==11 || e==12 ||
		e==13)
		return true;
		break;
	case 14:
		if (e==1 || e==2 || e==3 || e==4 || e==5 || e==6 ||
		e==7 || e==8 || e==9 || e==10 || e==11 || e==12 ||
		e==13 || e==14)
		return true;
		break;
	}
	return false;

}

zona vj(clasificador * clasif,imagen * img,unsigned char
porcentaje_niveles,unsigned char porcentaje_divisiones,unsigned
char porcentaje_escalas,  int * numEscalas_o, int *
num_escala_actual_o,float * escala_actual_o,  int *
num_divisiones_por_escala_o, int * division_actual_o, int *
tam_ventana_actual_o, bool * cancelar ) {
	int i,j,k;
	int e;

	float RLum_=RLum*8.225f;
	float GLum_=GLum*4.047f;
	float BLum_=BLum*8.225f;

	unsigned char lum [img->alto][img->ancho];
	for (i=0;i<img->alto;i++) {
			for(j=0;j<img->ancho;j++)
			{
			lum[i][j]=(unsigned char)((float)((img-
>img[i][j] & 0xF800)>>11)*RLum_+(float)((img->img[i][j] &
0x7E0)>>5)*GLum_+(float)((img->img[i][j] & 0x1F))*BLum_);

			}
		}

	unsigned int img_int[img->alto][img->ancho];
	for(i=0;i<img->alto;i++) //i=Y;j=X array[Y][X]
		for(j=0;j<img->ancho;j++)
		{
			if(i>0 && j>0)
			img_int[i][j]=img_int[i-1][j]+img_int[i]
			[j-1]-img_int[i-1][j-1]+lum[i][j];
			else if(i==0 && j>0)
				img_int[i][j]=img_int[i]
[j-1]+lum[i][j];
			else if(j==0 && i>0)
				img_int[i][j]=img_int[i-1][j]
+lum[i][j];
			else
				img_int[i][j]=lum[i][j];


		}
```

```
	zona cara = zona();
	cara.alto=0;
	cara.ancho=0;


	double sum_etapas;
	int xf,yf,wf,hf,x,y;
	int sum_caracteristica;
	int Rec;
	int pasoX,pasoY;
	int anchoCaracteristicaEscalado;
	int altoCaracteristicaEscalado;
	int spy;
	int spx;
	float escala;
	int h;
	int w;
	int paso;
	float anchoEscala= (float)img->ancho/(float)clasif->tamx;
	float altoEscala = (float)img->alto/(float)clasif->tamy;
	float escalaInicial;
	int escalasMaximas;
	int numEscalas;
	int nivelMaximo =(int)((float)clasif-
>etapas.size()*((float)porcentaje_niveles/100.f))-1 ;

	if(altoEscala < anchoEscala )
		escalaInicial =  altoEscala;
	else
		escalaInicial = anchoEscala;
	escalasMaximas =(int)((log(1.0f/(float)escalaInicial)/
log(EscalaActualizacion))+0.5f);
	numEscalas = (int)
((float)escalasMaximas*(float)porcentaje_escalas/100.0f);
	for(e=1;(int)e<=escalasMaximas;e++)
	{
		if(escala_admitida(numEscalas,e))
		{
			escala =escalaInicial*pow(EscalaActualizacion,
(int)e-1);

			w= (int)(clasif->tamx*escala);
			h = (int)(clasif->tamy*escala);
			if (escala>2) paso=(int)(escala-0.5f); else
paso=2;
			spx=(int)(((float)(img->ancho-w-1)/(float)paso
+0.999f)*((float)porcentaje_divisiones/100.f));
			spy=(int)(((float)(img->alto-h-1)/(float)paso
+0.999f)*((float)porcentaje_divisiones/100.f));

			if (spx>0 && spy>0)
			{

				*numEscalas_o= escalasMaximas*100/
				porcentaje_escalas;
				*num_escala_actual_o=(int)e;
				*escala_actual_o=escala;
				*num_divisiones_por_escala_o=spx*spy;
				*tam_ventana_actual_o=w;
```

```
            anchoCaracteristicaEscalado=(int)(escala*clasif-
>tamx);
      altoCaracteristicaEscalado=(int)(escala*clasif->tamy);

                        pasoX=(img->ancho-
            anchoCaracteristicaEscalado)/spx;
                       pasoY=(img->alto-
      altoCaracteristicaEscalado)/spy;

                        for(y=0;y<spy;y++)
                        {

                                    for(x=0;x<spx;x++)
                                          {

                                          *division_actual_o=x
+y*spx;

                                          if (*cancelar){
                                                cara.alto=0;
                                                cara.ancho=0;
                                                return cara;
                                          }


                                     for(i=0;i<(int)clasif-
                                    >etapas.size();i++)

      {



                                                sum_etapas=0.0;
                                    for(j=0;j<(int)clasif-
                                    >etapas.at(i).arboles.size();j++)

                                                {

                                                sum_caracteristica=0;
                                    for(k=0;k<(int)clasif-
                                    >etapas.at(i).arboles.at(j).caracte
                                    risticas.size();k++)

                                                {


                        xf=(int)(x*pasoX+(int)((float)clasif-
>etapas.at(i).arboles.at(j).caracteristicas.at(k).x*escala));

                        yf=(int)(y*pasoY+(int)((float)clasif-
>etapas.at(i).arboles.at(j).caracteristicas.at(k).y*escala));

                        wf=(int)((float)clasif-
>etapas.at(i).arboles.at(j).caracteristicas.at(k).ancho*escala);

                        hf=(int)((float)clasif-
>etapas.at(i).arboles.at(j).caracteristicas.at(k).alto*escala);

                        Rec=img_int[yf][xf]+img_int[yf+hf][xf
+wf]-img_int[yf][xf+wf]-img_int[yf+hf][xf];

                        sum_caracteristica=sum_caracteristica
```

```
+Rec*(int)clasif-
>etapas.at(i).arboles.at(j).caracteristicas.at(k).peso;

                    }


                    if((double)sum_caracteristica<clasif-
>etapas.at(i).arboles.at(j).umbral*(double)(w*h*255))

                          sum_etapas=sum_etapas+clasif-
>etapas.at(i).arboles.at(j).valor_izq;

                    else

                          sum_etapas=sum_etapas+clasif-
>etapas.at(i).arboles.at(j).valor_der;

             }


             if(sum_etapas<clasif->etapas.at(i).umbral) break;


             if (i==nivelMaximo)

                    {

                    cara.x=x*pasoX;

                    cara.y=y*pasoY;

                    cara.ancho=anchoCaracteristicaEscalado;

                    cara.alto=altoCaracteristicaEscalado;

                    cara.color=0xF800;

                    return cara;

                    }


                                                                  }

                                              }
                                  }


                    }
             }

      }

      return cara;

}
```

REFERENCIAS

[1]: Xilinx. http://www.xilinx.com/fpga/asic.htm.

[2]: National Instruments. http://sine.ni.com/cs/app/doc/p/id/cs-14155 .

[3]: Bittware. http://www.bittware.com/fpga-dsp-applications/applications-stories/ballistic-digitizer.

[4]: Bittware. http://www.bittware.com/fpga-dsp-applications/applications-stories/stealth-imaging-radar-system

[5]: E. Hjelmas and B.K. Low, “Face detection: A survey”. Computer Vision and Image Understanding, vol. 83, pp. 236-274, 2001.

[6]: M-H Yang and D.J. Kriegman, “Detecting faces in images: A survey”. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 1, pp. 34-58, 2002.

[7]: P. Viola and M. Jones, “Robust real-time face detection”. In Proc. Of IEEE Workshop on Statistical and Computational Thories of Vision, 2001.

[8]: Hayes, John P. Computer Architecture and Organization. New York: McGraw-Hill Publishing Company, 1988.

[9]: Xilinx. http://www.xilinx.com/products/silicon-devices/fpga/

[10]: Altera. http://www.altera.com/products/selector/

[11]: Standford University. http://www.stanford.edu/~hennessy/

[12]: Xilinx. http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[13]: Xilinx. “MicroBlaze Processor Reference Guide” http://www.xilinx.com/tools/microblaze.htm

[14]: Altera. http://www.altera.com/devices/processor/nios2/ni2-index.html

[15]: Altera. “Nios II Processor Reference Handbook” http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[16]: véase [14]

[17]: University of Texas, Austin. http://www.utexas.edu

[18]: IBM, “Understanding Static RAM Operation”, 1997

[19]: Steven K. Reinhardt, “Design of Microprocessor-based Systems”, 1999

[20]: véase [19].
[21]: José L. Martín González, “Electrónica Digital”. Ed. Delta Publicaciones, 2007
[22]: Terasic Technologies Inc. , “DE2-115 User Manual”, 2010
[23]: Altera. http://www.altera.com/education/univ/materials/boards/de2-115/unv-de2-115-board.html
[24]: Altera Wiki. http://www.alterawiki.com/wiki/UClinux.
[25]: véase [7].
[26]: Kwok-Wai Wong. University of Hong Kong. “http://www.cse.cuhk.edu.hk/~khwong/www2/cmsc5707/5707_6_face_detection.ppt”.
[27]: Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Eurocolt ’95, pages 23–37. Springer-Verlag, 1995.

[28]: Altera. Nios II Performance Benchmarks. http://www.altera.com/literature/ds/ds_nios2_perf.pdf
[29]: Xilinx. MicroBlaze Processor Performance Levels. http://www.xilinx.com/tools/microblaze.htm